Dr. Dobb's Journal

SOFTWARE TOOLS FOR ADVANCED PROGRAMMERS

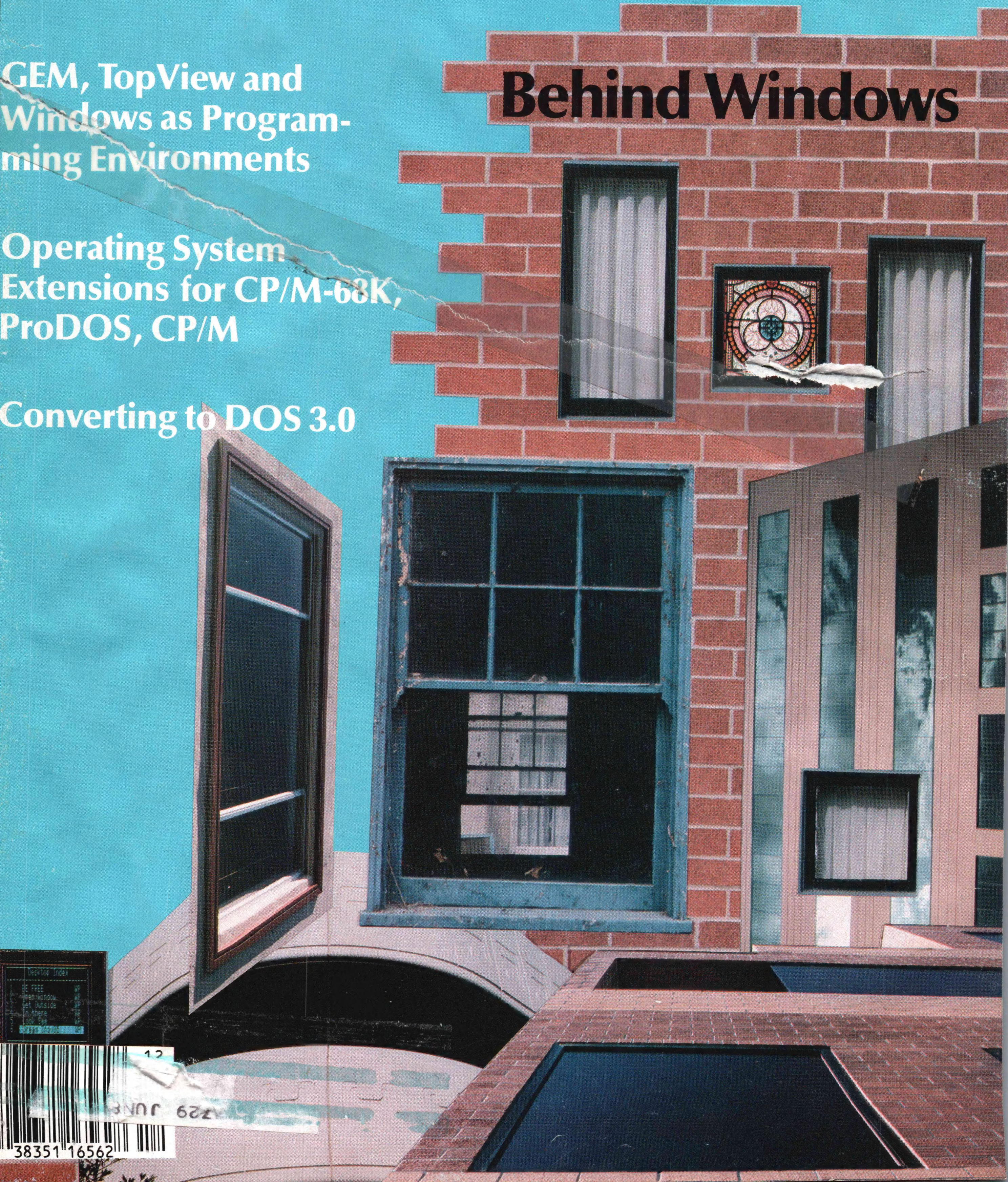
#110 DECEMBER 1985 $2.95 (3.95 CANADA)

Behind Windows

GEM, TopView and Windows as Programming Environments

Operating System Extensions for CP/M-68K, ProDOS, CP/M

Converting to DOS 3.0



Optotech, Inc.

The 5¼ inch Optical Disk Drive Is Here!

Optical Disk Drive 5984

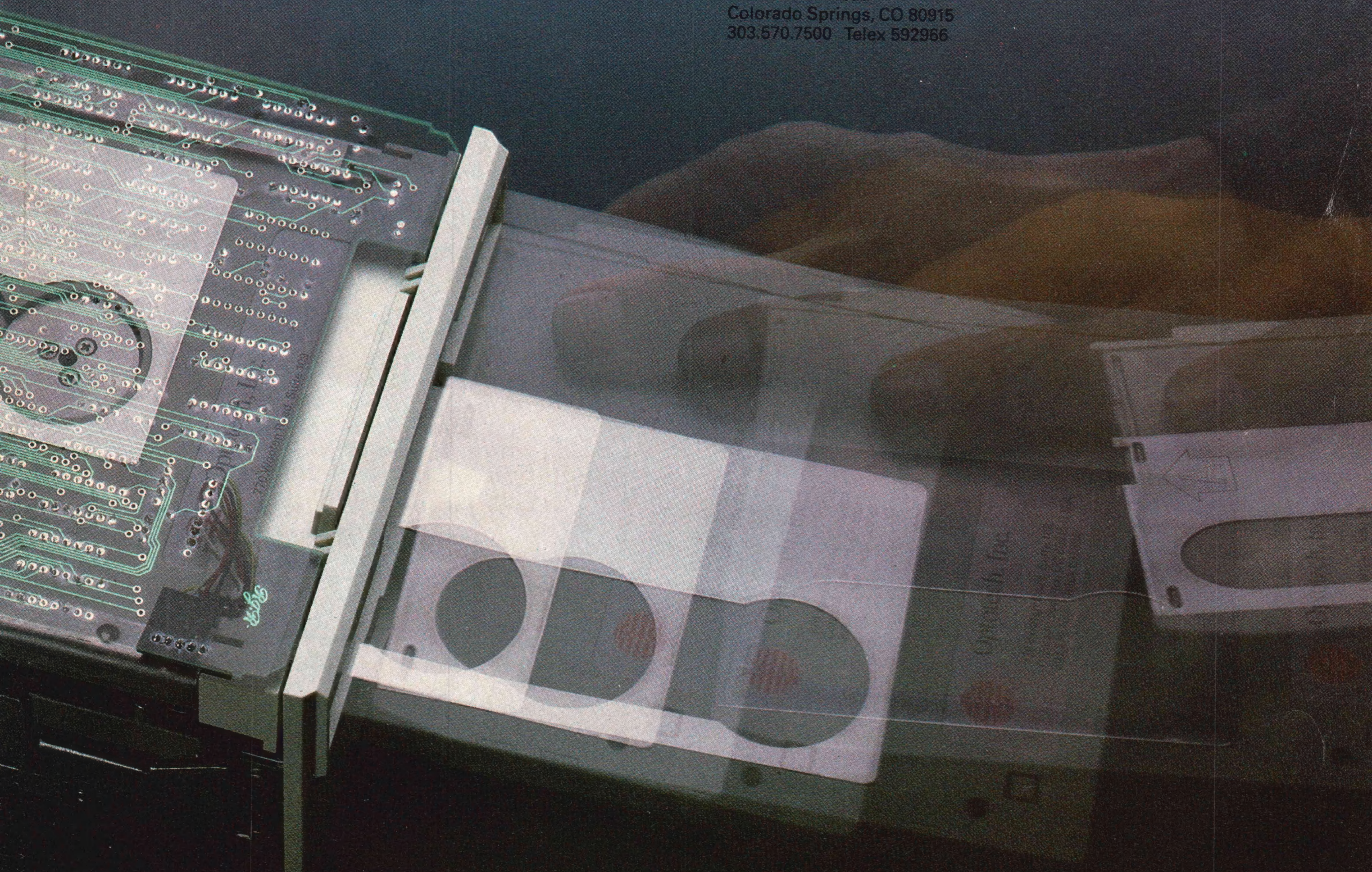
- *200 megabytes on a removable cartridge.*
- *Fast access read and write.*
- *For PCs and minis.*
- *Extensive interface software.*
- *Available Now.*

The Preferred Solution For:

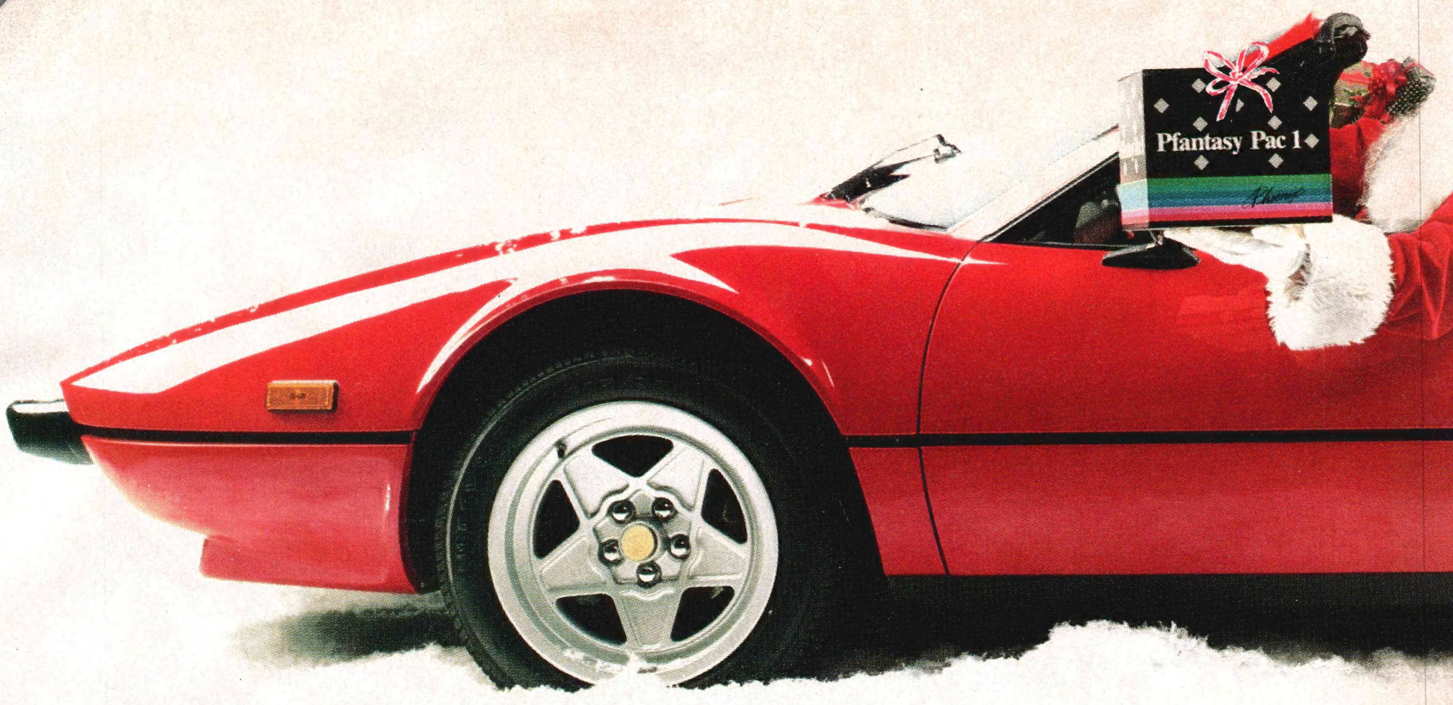
- *Database–large, portable, indelible and updatable*
- *Online Mass Storage–integral backup*
- *Imaging–capacity with removability*

Optotech, Inc.

770 Wooten Road
Colorado Springs, CO 80915
303.570.7500 Telex 592966



Circle **no. 124** on reader service card.



Buy Yourself A Pfantas
Just $~~12~~

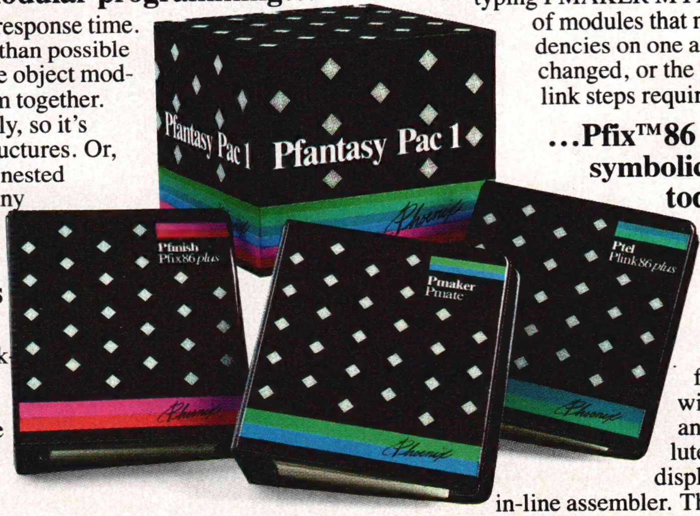
Give your favorite programmer a holiday treat – Phoenix Pfantasy Pac1. Six of the best-engineered, highest performance programming packages on the market today. $1800 sold separately. $1295 as Pfantasy Pac1 with new software, and a unique on-line software update service. All at the one-time holiday price of $995. The perfect gift for the knowledgeable programmer. You'll get:

...Plink™86 Plus, our new linkage editor, adds a new dimension to modular programming...

Cache overlays to reduce response time. Link programs much larger than possible with standard linkers. Merge object modules so you can package them together. Reload overlays automatically, so it's easier to design program structures. Or, create up to 4,095 overlays, nested up to 32 levels, in one or many files, on one or many disks.

...Plus, Pmate,™ the text editor that works the way you do...

You can run Pmate in background while you perform another task, or call it up instantly while you're inside another program. With Pmate, you can reassign any command or key, so you can use keystrokes learned on other editors or make your keyboard Dvorak. And, you can assign complicated editing tasks to a keystroke, so you never have to re-key them.



...Pmaker,™ the perfect program development manager...

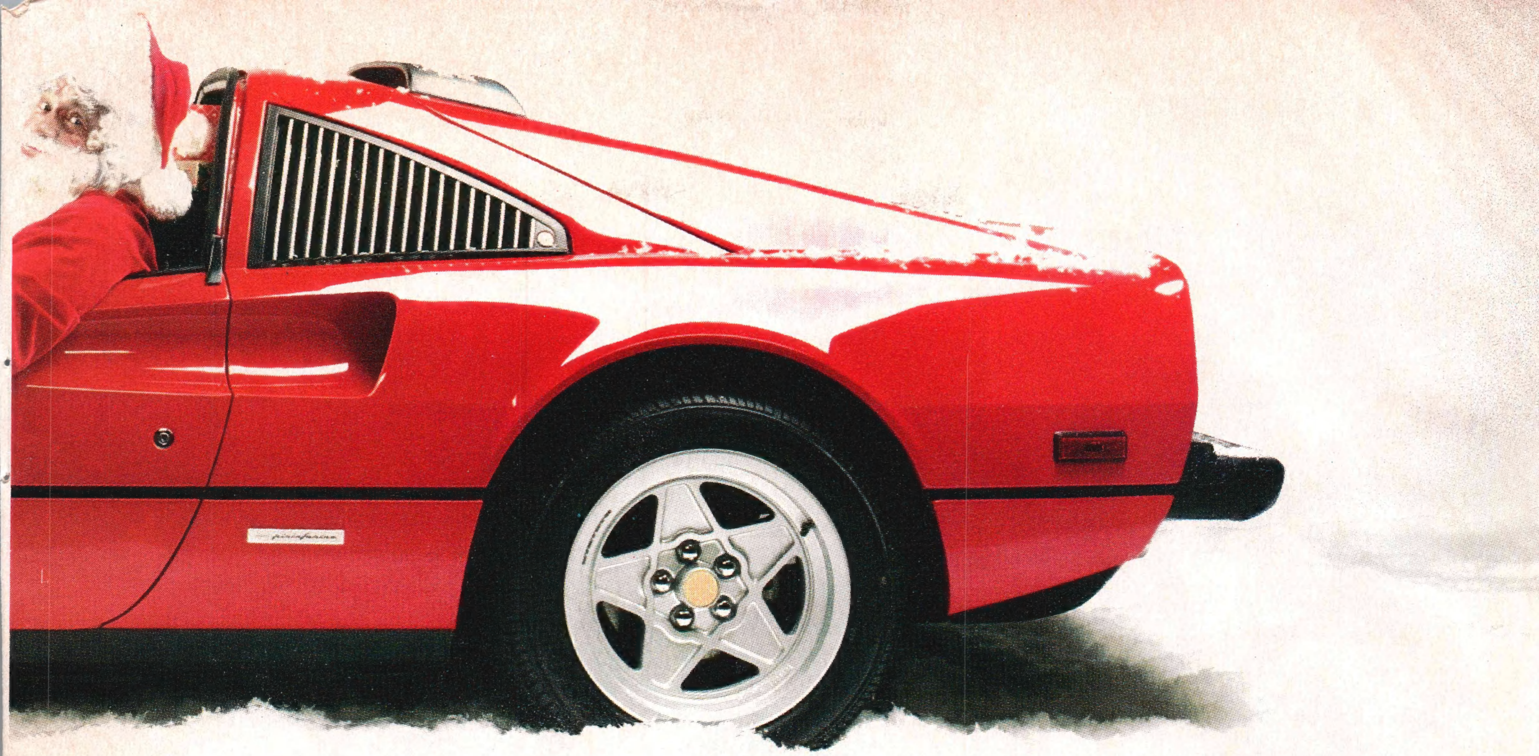
Like UNIX™ Make and MMF, Pmaker updates and tracks modifications in your programs. With Pmaker, generating a final copy of your program is never more complicated than typing PMAKER MYFILE, regardless of the number of modules that make up MYFILE, their dependencies on one another, the number of modules changed, or the variety of compile/assemble/link steps required to generate the program.

...Pfix™86 Plus, the most advanced symbolic debugger on the market today...

With its adjustable window display of source file, disassembled object, data area, stack, breakpoint settings, CPU and coprocessor registers, Pfix takes the guesswork from debugging. You can debug without a listing, since you can see and enter symbolic names or absolute addresses in breakpoints, data displays, expressions, or with the in-line assembler. The entire 80xx and 80xxx series of

*Price effective through December 31, 1985.

Circle **no. 139** on reader service card.



y Pac™ For The Holidays.

~~5.~~ $995.*

16-bit processors is supported, along with the user-assignable string and numeric variables, up to 100-step trace-back, and debug log to disk or printer. Breakpoints can be set in the source file window, whose display can be synchronized with the disassembled object during tracing; symbols can be added incrementally during the debug session; and the disassembly written to disk.

...Pfinish,™ the program that maximizes your program's efficiency...

Pfinish helps you turn your beta-test product into a software work of art. It analyzes your program or the entire operating environment during execution, and produces reports which tell you how much time was spent in each routine or interrupt, who called it, how many times, and much more. Wasteful, inefficient, or non-optimal areas of code become immediately apparent, whether they're in your code, the compiler library, or in the operating system routines themselves.

...Ptel™ gets the lead out of binary file transfer...

You get error-free file transfer and access to mainframes, minis, and micros. ASCII. XModem. Modem7. Telink and Kermit. MS-DOS® 2.x or 3.x. With Ptel. You can transfer 8-bit binary files over a 7-bit data path with Kermit. Masked file-name transfers with XModem. Or, transfer whole subdirectories with a single command using Telink. Ptel keeps track of the original file size and creation date. Ptel also offers a script language, backward scrolling, and the ability to handle DOS commands from inside the program.

...And, Phoenix's new on-line software update service. Free to all registered Pfantasy Pac1 owners.

Pfantasy Pac1 software works in MS-DOS/PC DOS environments, is completely language-independent, and is available for the IBM® PC, XT,™ AT,™ and compatibles, the Wang® PC, the TI Business-Pro,™ and the Tandy® 2000.

So, spoil yourself a little. Get Pfantasy Pac1, and make your favorite programmer's dream come true.

For more information contact: Phoenix Computer Products Corporation, 320 Norwood Park South, Norwood, MA 02062. Or call: (800) 344-7200. In Massachusetts (617) 762-5030.

***Programmers' Pfantasies™* by**

Phoenix

Dr. Dobb's Journal

Editorial
Editor-in-Chief *Michael Swaine*
Managing Editor *Frank DeRose*
Editorial Assistant *Sara Noah*
Contributing Editors *Robert Blum, Dave Cortesi, Ray Duncan, Allen Holub*
Copy Editor *Vince Leone*
Typesetter *Jean Aring*

Production
Production Manager *Bob Wynne*
Art Director *Shelley Rae Doeden*
Production Assistant *Alida Hinton*
Cover Artist *Tom Upton*

Circulation
Sub. Fulfillment Mgr. *Stephanie Barber*
Subscription Mgr. *Maureen Kaminski*
Book Marketing Mgr. *Jane Sharninghouse*
Single Copy Sales Mgr. *Stephanie Barber*

Administration
Finance Manager *Sandra Dunie*
Business Manager *Betty Trickett*
Accounts Payable Supv. *Mayda Lopez-Quintana*
Accounts Payable Asst. *Denise Giannini*
Billing Coordinator *Laura Di Lazzaro*
Accountant *Marilyn Brown*
Adm. Coordinator *Kobi Morgan*

Advertising
Advertising Director
Shawn Horst (415) 424-0600
Advertising Sales
Walter Andrzejewski (617) 567-8361
Lisa Boudreau (415) 424-0600
Beth Dudas (714) 643-9439
Michele Beaty (317) 875-8093
Systems Manager *Ron Copeland*
Administrative Manager *Anna Kittleson*

M&T Publishing, Inc.

Chairman of the Board *Otmar Weber*
Director *C.F. von Quadt*
President *Laird Foshay*

Dr. Dobb's Journal (USPS 307690) is published monthly by M&T Publishing, Inc., 2464 Embarcadero Way, Palo Alto, CA 94303, (415) 424-0600. Second class postage paid at Palo Alto and at additional entry points.

Address correction requested. Postmaster: Send Form 3579 to *Dr. Dobb's Journal*, P.O. Box 27809, San Diego, CA 92128. **ISSN 0278-6508**

Customer Service: For subscription problems call: outside CA 800-321-3333; within CA 619-485-9623 or 566-6947. For book, back issue, or disk order problems call 415-424-1474.

Subscription Rates: $25 per year within the United States. Foreign subscription rates: $52.00 for airmail, $42.00 for surface mail. Foreign subscriptions must be pre-paid in U.S. Dollars, drawn on a U.S. Bank.

Foreign Distributor: Worldwide Media Service, Inc., 386 Park Ave. South, New York, NY 10016, (212) 686-1520 TELEX: 620430 (WUI)

People's Computer Company

Dr. Dobb's Journal is published by M&T Publishing, Inc. under license from People's Computer Company, 2682 Bishop Dr., Suite 107, San Ramon, CA 94583, a non-profit, educational corporation.

December 1985
Volume 10, Issue 12

CONTENTS

In this Issue

This is our second annual operating systems issue. We've tried to present a variety of tools for extending the performance of several operating systems. We looked for techniques and programs that would make software development easier: a debugging tool, techniques for expanding the available memory for programs, a conditional instruction tool for CP/M-68K.

We also took a look at three commercial products that claim to extend the performance of the MS DOS operating system: DRI's GEM, IBM's TopView, and Microsoft's Windows. We talked to developers of each product, asking what programmers need to know to develop applications compatible with these operating environments and why a programmer might want to do such a thing.

Look closely at the issue next month; it won't look quite the same. Our art director has introduced some changes inside and outside the magazine that we think you'll like.

We don't think you'll like hearing that Dr. Dobb's Clinic is closing with this issue. We don't like telling you. The resident intern, Dave Cortesi, has gone off to pursue interests having nothing to do with software. We'll miss Dave's gentle wit. We'd like to reopen the Clinic, but we haven't found anyone yet to fill Dave's shoes, and until we do, we won't.

Next month: 68000 programming.

STATEMENT OF OWNERSHIP, MANAGEMENT, AND CIRCULATION

(Act of August 12, 1970, Section 3685, Title 39, United States Code)

1. Title of Publication: Dr. Dobb's Journal, Publication No. 307690.
2. Date of Filing: October 12, 1985.
3. Frequency of Issue: Monthly (12 issues, $25).
4. Location of Known Office of Publication: 2464 Embarcadero Way, Palo Alto, CA 94303.
5. Location of Headquarters of General Business Offices of the Publishers: 2464 Embarcadero Way, Palo Alto, CA 94303.
6. Names and Addresses of Publisher, Editor, and Managing Editor: Publisher, Laird Foshay, 2464 Embarcadero Way, Palo Alto, CA 94303. Editor, Michael Swaine, 2464 Embarcadero Way, Palo Alto, CA 94303. Managing Editor, Frank DeRose, 2464 Embarcadero Way, Palo Alto, CA 94303.
7. Owner: Otmar Weber, 2464 Embarcadero Way, Palo Alto, CA 94303; and C. F. Von Quadt, 2464 Embarcadero Way, Palo Alto, CA 94303.
8. Known Bondholders, Mortgages, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages or Other Securities: Markt & Technik, Hans-Pinsel-Strasse 2 8013 Haar bei Munich W. Germany.
9. Extent and Nature of Circulation:
   A. Total number of copies printed: Average number of copies each issue during the preceding 12 months: 41,693. Actual number of copies of single issue published nearest to filing date: 45,037.
   B. Paid Circulation. 1. Sales through dealers and carriers, street vendors, and counter sales. Average number of copies each issue during preceding 12 months: 8,707. Actual number of copies of single issue published nearest to filing date: 7,815. 2. Mail subscriptions. Average number of copies each issue during preceding 12 months: 26,568. Actual number of copies of single issue published nearest to filing date: 30,417.
   C. Total Paid Circulation. Average number of copies each issue during preceding 12 months: 35,275. Actual number of copies of single issue published nearest to filing date: 38,232.
   D. Free distribution by mail, carrier, or other means, samples, complimentary, and other free copies. Average number of copies each issue during preceding 12 months: 2,353. Actual number of copies of single issue published nearest to filing date: 3,305.
   E. Total distribution. Average number of copies each issue during preceding 12 months: 37,628. Actual number of copies of single issue published nearest to filing date: 41,537.
   F. Copies not distributed. 1. Office use, left over, unaccounted, spoiled after printing. Average number of copies each issue during preceding 12 months: 150. Actual number of copies of single issue published nearest to filing date: 150. 2. Returns from news agents. Average number of copies each issue during preceding 12 months: 3,915. Actual number copies of single issue published nearest to filing date: 3,350.
   G. Total. Average number of copies each issue during preceding 12 months: 41,693. Actual number of copies of single issue published nearest to filing date: 45,037.

I certify that the statements made by me above are correct and complete.

Laird Foshay, Publisher

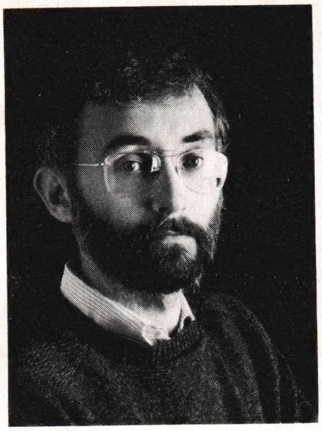
Dr. Dobb's Journal

ARTICLES

COLUMNS

DEPARTMENTS

EDITORIAL



It's been a litigious autumn in the software industry. At press time, Apple had forced DRI into changing the GEM Desktop; Apple lawyers were eyeing Atari and Microsoft; and Atari, anxious to get its software into ROM, was reportedly ready to challenge Apple. Apple's contention that its copyright protected not only the expression of algorithms in code but also the visual expression of the desktop (and implicitly that it had invented that desktop) had yet to be tested in court.

Adapso had been running ads attempting to educate software users about the criminal nature of unauthorized copying of software. Several software publishers had brought suit against customers to aid in the pedagogical process—all in reaction to a perceived loss of (potential) revenues and all very reminiscent of a campaign by the author of MITS BASIC a decade earlier. It wasn't a good tactic then to call your (potential) customers thieves and probably isn't now, but how are authors' and publishers' investments in software to be protected without alienating users?

Vault, a manufacturer of a copy-protection product, concluded that a Canadian company must have illegally disassembled Vault's code in order to produce its unlocking product; so Vault found a state in which it thought it could get a decision in its favor and sued. Vault lost, but one decision in a Louisiana court hardly dissipates the moral ambiguity that surrounds the entire copy-protection issue. Do we really want this escalating arms race between protectors and crackers?

The Vault case dealt with trade secrets. Copyright applies to works that are in some manner made public. No author of a book would object to someone analyzing his work in order to understand it—disassembling it, if you will. Trade secrets, on the other hand, are necessarily secret. Whom do trade secrets protect, and whom does copyright protect? Many software publishers treat the two protection methods as compatible, but does it really make sense that you can protect as a trade secret something protected under copyright law? As a book reviewer I could, under the fair-use interpretation of copyright law, excerpt and analyze a work of literature. Could I, as a software reviewer, disassemble and publish portions of commercial programs to show the workings of key algorithms?

These questions suggest, at the very least, that not everyone is of one mind regarding the protection of intellectual property. They also suggest something more alarming: that precedents are being set and opinions are being molded by bluster and threat, by cases that never come to court, and by venue-choosing ploys that take advantage of differing levels of understanding of the issues in different states. It shouldn't be that way.

You could do something about it. You probably know more about the issues than do most journalists, judges, or jurors. And I suspect you may have more empathy for the user than does the average software publisher, who often has stockholders to answer to. I urge you to make your expertise available to those making decisions with long-term implications. Educate lawyers and editors.

Become an expert witness for rational software law.

Michael Swaine

Michael Swaine

Small-C

Dear *DDJ*,

The Small-C problem reported by Gregor Owen (Letters, July 1985) has already been identified, fixed, and reported to Small-C users on my mailing list. It also appears in the "Small-C Update" article (*DDJ*, August 1985).

As it turns out, Owen's change does not address the real problem and introduces serious problems of its own. It is generally not a good idea simply to remove functions from a program even if they are referenced only once. Most programmers have reasons, other than the introduction of bugs, for placing even poorly documented functions in programs.

The function result( ) serves the vital role of causing the expression analyzer to note for binary operators whether or not the result is an address, and if so whether it is a character or integer address. Removing it will cause the analyzer always to assume the attributes of the left operand for the operator value. For example, "addr1 − addr2" should yield an integer result indicating the number of objects (bytes or words) between the two addresses. Without result( ), Small-C will think it has another address, and that has ramifications for the further process of expression evaluation.

The real problem was that the analyzer was losing track of the attributes of the lvalue for ?= operators when the right side evaluated to an address. The correction was to make hier1( ) (in CC31.C) remember two of the lvalue attributes for use at store time. This is done as follows:

1. define the local integer array lval3[2] after lval2[8].
2. between "if(k==0) . . ." and "if(lval[1]) . . ." insert
```
lval3[0] = lval[0];
lval3[1] = lval[1];
```
3. change the call to store( ) to read
```
store(lval3);
```

Owen's change appeared to work because in his test case the error he introduced caused the compiler to sidestep the original problem. The troublesome statement was:

```
1ptr += r - bytes
```

where "r" was an integer and "bytes" was an array name (yields the array address). As mentioned above, the removal of result( ) would cause "r − bytes" to yield an integer result, thereby avoiding the original problem.

By the way, one might question the meaning of an integer minus an address. K&R point out on page 99 that such expressions are illegal.

Jim Hendrix
Rt. 1, Box 585
Oxford, MS 38655

Information Age Issues

Dear *DDJ*,

Please continue publishing articles like Dean Gengle's "Information Age Issues" (June 1985). Such ethical issues are too important to leave in the hands of a systems designer.

I must, however, take issue with Gengle's comment that "no one is talking about our rights to access information that is *not* private and that was collected legally at taxpayers' expense. Census data and Library of Congress information come to mind."

Your readers should know that both the American Library Association Government Documents Roundtable and the Special Libraries Association Government Relations Committee have indeed been talking about it. In addition, the Joint Committee on Printing of the Congress, which oversees the Government Printing Office, has been very concerned that material produced in machine-readable form by Federal agencies has not been made available to the GPO for distribution to the public through the depository libraries because it is not "printed."

Librarians protested strongly the distribution of census data in microfiche-only format because such data is not readily accessible to the poor or illiterate. Distribution on magnetic tape is a great service to those who can afford to have the tapes processed but may harm those who depend on such data for distribution of entitlement program benefits.

Write to your representatives and and urge them to support the efforts of the Joint Committee on Printing, the ALA, and SLA.

I'm not sure what Gengle is referring to by "Library of Congress information." He may be suffering from the misinformation provided by an episode of the television program "Whiz Kids" in which the elfin heroes broke into the Library of Congress computers and did a homework assignment. I doubt if there are any files at the LC, or anywhere else for that matter, that could produce the results obtained. In fact, the files at the LC are bibliographic, containing references to literature, but not full texts—at least not yet. Not only are these bibliographic files available to the public at the LC but they are available on magnetic tape to libraries and other institutions throughout the world. Many libraries located in this country access those records through OCLC, a bibliographic utility with over 11 million records. Some agencies do apparently try to hide

Wendin® puts Power and Punch in your PC

ORDER NOW! Prices on PCUNIX™ and PCVMS™ go up to $99 effective January 1, 1986

Operating System Toolbox™ only $99.00

Operating System Toolbox™ is a software construction set that enables you to build your own multitasking, multiuser operating system. All you need is your creativity and imagination to write a shell and link it with the Toolbox. What you receive for your effort is an operating system that interacts with the way **YOU** work, performs the jobs that **YOU** need in the way **YOU** want them done. Toolbox runs MS-DOS and PC-DOS programs with full source code written in Microsoft "C", organized into seven basic modules, all on disk. The package includes both source and object code for each module (just in case you don't have Microsoft "C"), plus a fantastic, in-depth, step-by-step instruction manual on how to build your personal operating system.

The best part of the news is the price — so low, you'll want to order today, just to keep up with the state of the art in operating systems.

PCUNIX™ only $49.00

PCUNIX™ is powerful! It is a true multitasking, multiuser operating system similar to the popular UNIX® operating system from AT&T. This first release includes 30 utilities, designed to make your program development a snap.

PCUNIX™ is versatile! It runs MS-DOS and PC-DOS programs, so that you'll never have to buy another set of development tools! Simply use your existing compilers and linkers. You need nothing more! As with the development of all of our Operating Systems, Wendin has used the Wendin Operating System Toolbox™ to construct PCUNIX™. Because of this, your programs will have access to the over 70 enhanced system services found in the Wendin Operating System Toolbox!™

PCUNIX™ is complete! Full source code to the shell and all utilities, written in "C", is included with this incredible package. And if you would like the source to Wendin's Personal Operating System Kernel, you may also want to purchase the Wendin Operating System Toolbox.™

PCVMS™ only $49.00

PCVMS™ is Wendin's version of the popular VAX/VMS operating system, which was developed by Digital Equipment Corporation for their line of VAX computers. PCVMS™ turns your IBM-PC, XT, AT, or true compatible into a supercharged, multitasking, multiuser workstation that runs MS-DOS and PC-DOS programs.

As with the development of all of our Operating Systems, Wendin has used the Operating System Toolbox™ to construct PCVMS, allowing your programs access to the over 70 enhanced system services available in the Toolbox.

PCVMS™ comes with full source code to the PCVMS™ shell and its utilities, all written in "C". If you also want the source to Wendin's Personal Operating System Kernel, you may want to purchase the Wendin Operating System Toolbox.™

XTC® only $99.00

XTC® is the world's first multitasking editor for the IBM-PC. It also runs on IBM-XT and IBM-AT computers, as well as true compatibles. Designed BY programmers FOR programmers, XTC is the ultimate editing tool for software developers using C, PASCAL, ASSEMBLY, BASIC, FORTRAN, and other languages.

Why is XTC® the ultimate tool for editing in your development environment? Because it has powerful features like MULTITASKING MACROS, WINDOWS, TEXT BUFFERS, UNDO N TIMES, MACRO PROGRAMMING CONTROL STRUCTURES and VARIABLES, as well as blinding speed that leaves other editors in the stone age.

XTC® comes with full source code written in Microsoft Pascal, on 2 DSDD disks.

ORDER HOTLINE
509/235-8088
Master Card & Visa

Foreign orders inquire about shipping.
Domestic orders add $3.50/1st item, $1.00 each additional item for shipping, handling and insurance.

Washington residents add 7.8% sales tax.

WENDIN®
BOX 266
CHENEY, WA 99004

The people who make quality software tools affordable.

DEALER INQUIRIES WELCOME!

PC-DOS is a trademark of IBM
MS is a trademark of Microsoft
Unix is a trademark of AT&T
VAX/VMS is a trademark of Digital Equipment Corporation

Wendin and XTC are Registered Trademarks of Wendin, Inc.
PCVMS, PCUNIX, and Operating System Toolbox are Trademarks of Wendin, Inc.

Circle no. 112 on reader service card.

their information from the public, but the Library of Congress certainly does not.

Bruce B. Cox
Documents Librarian and
Automation Committee
Linda Hall Library
5109 Cherry Street
Kansas City, MO 64110

C Compiler Review

Dear *DDJ*,
I read with great interest your August 1985 review of MS DOS C compilers. I have two comments I would like to interject into the debate.

First, regarding the review of the documentation—this is always a touchy area, being rather subjective in impact—I am not convinced that a C compiler manual should include what is essentially a rehash of the standard K&R syntax chapters nor a rehash of the standard Unix manual's chapters 2 and 3. That is, unless a compiler and run-time library deviate from these accepted standards, I see no reason to repeat them. It only serves to confuse. As an example, I find the Lattice C documentation difficult to use because I generally expect it to be similar to the Unix manual. It isn't, and I always get lost trying to find something.

It is much better simply to include a "pointer" to the appropriate documentation with the caveat that any functions and/or syntactic notes included in the product's manual imply a deviation from the norm. Sort of like a "search path" if you will. One uses the Unix and K&R manuals unless the product's manual overrides them.

Second, the authors did not include as a criterion of judgement the number of other products that support a particular compiler. For instance, although Lattice C does not necessarily produce the tightest code nor run the quickest, it is important that one can obtain interfaces to major products. The Oracle DBMS includes a host-language interface (HLI) to Lattice C, and a HLI to dBASE II and III can be obtained from Lattice. Such interfaces are of great significance, and in these systems the choice of compiler will hinge entirely upon that single criterion. It is better, in other words, to put up with slightly slower execution than to have to generate in house an entire nontrivial HLI.

Otherwise, I was pleased with the review—it was obviously a strong effort. It is gratifying to this C and Unix nut to see the wide interest in these tools for programmers.

Brian Jay Wu
P.O. Box 203
Newbury Park, CA 91320

Dear *DDJ*,
When I heard that your magazine was going to do a review of C compilers and interpreters, I couldn't wait to read it.

The cover for the August 1985 issue claims that the review of C compilers is definitive. I think the review is marginally useful and generally misleading. Information important to the potential user was omitted. A list of considerations relating to the usefulness of the various C compilers follows:

1. Good luck to anyone who purchases the Microsoft compiler Version 3.0 without a hard disk! The four passes of the compiler total 290K, the CL.EXE driver takes 27K, the linker takes 41K, and the libraries needed total at least 120K. Add it up and you get a whopping 478K.

It is becoming common to run a compiler from inside your editor.

If you're tired of waiting, you're using the wrong file manager.

Be fast. Btrieve.™

If batch jobs and reports turn waiting time to nighttime, then wake up! You're using the wrong file manager.

Btrieve™ file management hates waiting as much as you do. It's written in assembly language especially for the IBM PC. And based on b-tree file indexing, with automatic balancing for access speed that won't degrade as your database grows. With Btrieve, your applications always run fast. So you'll be out the door faster.

The standard for networking. When your application requires multi-user file sharing, Btrieve/N (network version) sets the standard for the industry's most popular LANs: IBM's PC Network, Netware, Omninet, PC Net, Ether Series, Nestar, and NetOne. Btrieve/N offers *safe* network file management that coordinates simultaneous updates and prevents lost data.

Automatic file recovery. Btrieve provides automatic file recovery after a system crash, so accidents and power failures don't turn into database disasters. Your Btrieve data always comes back intact.

Fully-relational data management. Using SoftCraft's entire family of products gives you a complete, fully-relational database management system. Rtrieve™ adds report writing capabilities for generating the reports you need. Xtrieve™ speeds users through database queries with interactive, on-screen menus—no command language or special syntax.

For professional programmers. Btrieve is the fast, reliable answer for all your application development. In any development language—BASIC, Pascal, COBOL, C, FORTRAN, and APL. With multikey access to records. Unlimited records per file. Duplicate, modifiable, and segmented keys. With Btrieve, you can develop better applications faster. And know they'll run—*fast*.

Now available for XENIX

Suggested retail prices: Btrieve, $245; Btrieve/N, $595; Xtrieve, $195; Xtrieve/N, $395; Rtrieve, $85; Rtrieve/N, $175. Requires PC-DOS or MS-DOS 1.X, 2.X, or 3.X.

Btrieve, Xtrieve, and Rtrieve are trademarks of SoftCraft Inc.

SoftCraft Inc.

P. O. Box 9802 #917
Austin, Texas 78766
(512) 346-8380 Telex 358 200

Circle **no. 113** on reader service card.

COMMAND.COM needs about 18K and a programmer's editor may need 100K or more. We haven't even started to include a debugger, assembler, other libraries, object modules, or header files. In general, it would be nice to know the amount of space needed by the compiler, libraries, linker, and other assorted software the compiler needs to run.

2. The object-module format of the compilers was not mentioned. Microsoft compatibility may be an important issue for some people, especially those who buy libraries.

3. The assembler that comes with the Mark Williams compiler does not use Intel mnemonics.

4. The review mentions that the large model for Lattice allows disabling of pointer range checking, which avoids a wraparound on a 64K boundary when doing pointer arithmetic. Nowhere was it mentioned how perilous such an undertaking can be on a project of any size!

5. Several of the compilers support in-line assembly, which is extremely useful. I did not see any mention of the DeSmet or DRI compilers supporting that feature.

6. The DeSmet debugger, an excellent tool, wasn't even looked at!

7. The DeSmet compiler doesn't support the large memory model but does come with an overlay linker.

8. Although it can be useful to look at the size of the .EXE file produced by a compiler and linker, sometimes you want to know the sizes of the code and data produced by the compiler as well. Will the compiler tell you those statistics?

9. The reviewers didn't mention tradeoffs between compile/link time vs. speed of executable code (using an optimizer pass, for example).

10. The fact that Lattice and Computer Innovations automatically provide run-time stack-overflow checking was not mentioned.

I would like to have seen more of an attempt to separate out evaluations of the compilers from the associated libraries. I couldn't care less about the performance of libraries, since for applications that require speed and efficiency I write my own routines.

I have a second reason for writing. It seems to me that people deciding which compiler to choose might benefit from feedback from programmers actually using C compilers. Since C compilers are used in a wide variety of applications and environments under MS DOS, readers would likely get different kinds of information than a reviewer would provide. Why not allocate space for a running column where readers can sound off about their favorite compiler and give reasons why they like it?

If you readers like that idea, write to *DDJ* about the compiler you like to work with, mentioning aspects that haven't been covered. *[We would welcome this input.—Ed.]* Include your work environment (network, single-user, etc.); machine; any idiosyncracies or work requirements on programming style; and why the compiler you work with is the greatest thing since *DDJ*.

Tom Hogan
12801 Frost Rd.
Kansas City, MO 64138

DDJ

DR. DOBB'S CLINIC

by D. E. Cortesi

Clinic to Close

The Clinic opened for business in May 1981. Now, four and one half years later, career changes compel me to close it down. The landlord will soon install new and interesting enterprises on the premises, but I thought I'd use our last meeting to look back and say thanks.

Two things come to mind when I reflect on the column. The first is trivial: that damned editorial We. I can't say why I hobbled myself with that foolish affectation in the first column nor why I didn't drop it in the second, but I did and then I didn't and from then on I felt compelled to continue an established style.

The strongest impression I retain is of the quality of the *DDJ* readership. As originally conceived, this column was to be written mostly by its readers with the Intern editing and supplying continuity, like a radio talk show in slow motion. It never quite worked out; you readers never provided enough original material to make a column.

But you responded when your interest was caught—my goodness how you responded! I posed Don Taylor's highly abstruse problem in the mathematics of graphics and a dozen of you sent detailed, multipage discussions heavy with matrix notation. I put up David Ross's BASIC program and 20 of you sent line-by-line discussions of what it did and how to do it better. Although the column was never written by readers in the way originally planned, you did supply a lot of material, correct a lot of my mistakes, and lend me your enthusiasm over the years.

Many of those who wrote were made into co-columnists and quoted in the various issues by name. They are named once more, with the issues they graced by their presence, in the following list. For every one of those mentioned there were often two, three, or a dozen more who had the same insight but wrote later or in a less quotable way. My personal thanks to all who wrote—it's been a privilege to associate with such a classy bunch. You folks keep up the good work, you hear?

Allison, Dennis	8/81, 10/81
Amelang, Loren	2/83
Asay, Matt	10/81
Ashley, Allen	7/81, 10/81
Backus, Rex	4/85
Bacon, Phillip	7/81
Baenziger, Peter	8/84
Barker, L.	7/81, 10/81, 11/81, 10/82, 10/83
Barker, David	11/85
Barr, Michael	5/84, 8/85
Bates, Dan	9/81
Bećus, Georges A.	9/81
Behler, Alan	5/83, 11/83
Bomberger, Alan	10/83
Brickell, Gavin	10/82
Briggs, R. C.	8/84
Brown, Russel	8/81
Butler, Edmund	7/81
Cage, Thomas W.	2/85, 6/85
Caldwell, Taylor	8/81
Canniff, Paul	4/85
Carr, Thomas E.	9/81
Carter, Everett	8/85
Castle, Pat	2/85
Chrapkiewicz, Thomas	8/84
Clark, David	2/84
Crowfoot, Norman	9/81
Dale, Gary	1/85
Davis, Charles	5/85
Davis, M. Patton	10/81
Dayhoff, Edward S.	11/81
Dempsey, J.	3/83
Dinsmore, Perry	11/84
Dodd, Michael S.	11/85
Driscoll, Michael	9/81
Dugan, Dan	4/82
Edgar, Gerald	7/81
English, Glenn	11/85
Enterline, Tom	8/84
Falk, Richard	3/84
Farmer, Wayne	8/81, 10/81
Fay, Jack	11/85
Feign, David	8/84
Floyd, Chet	9/84, 2/85
Fordham, Malcom	4/84
Fritz, Randolph	5/83, 6/83
Gardner, Calvin L.	8/84
Glassey, C. Roger	12/81
Goldman, Oscar,	5/84
Goodman, Bob	7/81
Grant, Steve	8/84
Greenlaw, Richard	10/81
Gunn, John Paul	2/85
Hahn, Harvey	5/83
Hammond, Nick	10/82, 6/83, 10/83, 10/85
Head, Gene	12/81
Hills, Terence	11/81
Hoats, David L.	12/81
Hoffman, Lee H.	2/85
Hole, Dr. William T.	2/85
Hoolko, Bob	7/81
Howell, Edgar	4/82
Howell, Jim	5/82, 2/84, 4/85
Hutchinson, Aubrey	2/83–6/83
Hutchinson, W. G.	4/82
Irving, Dwight	11/83
Jackson, Terry	2/85
Jensen, John Thayer	6/83
Johnson, Bruce R.	2/85
Johnston, Bill	12/81
Jones, James	4/85
Kellogg, David	6/83
Kendall, Bruce	8/82
King, David	2/84
Knipp, Ernest	10/82, 2/83
Koehler, W. B.	11/83
Komusin, Bruce	4/82, 5/82
Lane, Tom	7/82
Laurino, James	1/82, 7/83
Leonard, M. David	7/81
Levy, David	7/82

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 190.**

A New Shell for MS DOS, Part 1: IBM Cursor Control and an Fgets that Edits

by Allen Holub

Back in March we presented a new root module for the CP/M-80 version of the Aztec C compiler. Since then, several people have written to say that the proper place for wildcard expansion, pipes, redirection, and the like is in the shell (i.e., the command interpreter), not in the C root module. There's no point in bloating the size of your programs when you don't have to. I sympathize with this attitude, but frankly, I don't want to write a new CCP for CP/M. Digital Research just makes this process too hard.

MS DOS, however, is another matter. The command interpreter, command.com, is just a program and can be replaced with any other program. Moreover, the DOS interface library provides much more in the way of system level functions than does CP/M. Most important, DOS Version 2 and higher supports an exec function that allows one program to load and execute a second program.

Because most of my gripes about MS DOS are really aimed at the behavior of command.com, it seemed to me that writing a new shell for MS DOS would be more productive than complaining for the rest of my life about the features I didn't have. C is, after all, a language remarkably well suited for writing operating systems. So, over the next few months we will look at a new shell for MS DOS.

I've implemented most of what I like about the the Unix C and Bourne shells: command line wild card expansion, redirection, history, shell variables, a reasonable language for writing shell scripts (.bat files under MS DOS) and the lot. I've also taken the opportunity to add a couple of features that I've often wished were present in the Unix shells, such as an interactive command line editor.

String Routines

We'll start looking at the shell itself next month. This month we're going to look at several support routines. As usual, these are all designed to be useful in their own right. They make up five sets. The first three listings are routines for string manipulation. They are small, but useful. Next( ) (Listing One page 19) gets from a string the next object separated by a delimiter. Its calling syntax is:

```
char *next( linep, delim, esc )
char **linep;
int delim, esc;
```

Linep is the address of a pointer that points at the first character of the string to be processed. Delim is the delimiter that separates objects. Esc is an escape character. The routine skips leading white-space (a space or tab) and remembers the position of the first non-white character. It then looks for a delimiter not preceded by an escape character and replaces the delimiter with a '\0.' It returns a pointer to the remembered non-white character and it modifies *linep to point to the first character beyond where the delimiter used to be. It won't modify *linep to point past the end of the string. If delim is a white character, leading white space won't be skipped.

Listing Two (page 19) and Listing Three (page 19) are two routines for copying. Cpy(dest,src) is functionally the same as strcpy( ). It copies the source string (src) to the destination string (dest). There is, however, one difference: whereas strcpy returns its first argument, cpy returns a pointer to the NULL at the end of the destination string. This allows us to add onto the destination string without first having to scan through the string to find its end. The second copy routine is cptolower(dest,src). Cptolower works just like cpy( ), except all uppercase characters are mapped to lower case as they are copied.

Moving the Cursor and Writing Characters

Listing Four (page 20) is a collection of low-level routines for cursor manipulation and character output for the IBM PC. These routines use direct ROM BIOS calls, and so are much faster than the usual output routines, which use the normal DOS functions[1]. Moreover, they let you move the cursor around without having to install the ANSI.SYS driver, which is excruciatingly slow. IBM has been very good about maintaining the ROM BIOS interface over different versions of the operating system. As long as you access the BIOS via the correct interrupt mechanism, the routines are portable.

I've used the int86( ) function from Lattice to generate the system video interrupt. Microsoft C has an identical function, so these routines will port to Microsoft C without any problems. The dos( ) function printed in this column in July 1985 can also be modified quite easily to generate a video interrupt. Just replace the INT 21H instruction on line 68 with an INT 10H instruction.

Listing Four contains the following routines:

getpage( )

This routine gets the currently active video page. When you are in text mode, several pages are available for writing, though only one is displayed at any given time. Getpage( ) returns the number of the page now being displayed. This information is needed by some of the other routines.

cursize(top, bot)
This routine adjusts the cursor size. Every character is represented on the display by a fixed number of scan lines. Cursize( ) causes the cursor to extend from the top scan line indicated to the bottom line indicated. For example, the standard monochrome display uses 14 lines numbered 0–13, the color adapter uses 8, numbered 0–7. So, on the color adapter, a normal underline cursor can be set up with the call cursize(6,7). You can make a large block cursor with the call cursize(0,7). Cursize(7,0) creates a two-part cursor, one line over the letter and a second line under the letter. Cursize(8,8) will make the cursor disappear.

posn = gcur(pagenum);
int pagenum;
short posn;
This routine returns the position of the cursor on the indicated video page. The position is returned in a single 16 bit short; the row number is in the top byte and the column number is in the bottom byte.

scur(posn, page)
short posn;
This modifies the cursor to rest at posn, which may be a value returned from gcur( ).

posn = getcur( );
setcur(posn);
These routines work just like scur( ) and gcur( ) except that they access the page currently being displayed.

wchar(c)
int c;
This routine writes a single character to the screen. It is dramatically faster than putchar( )—at least the one that comes with the Lattice compiler. Wchar( ) moves the cursor as it prints, just like putchar( ). However, the only control codes it recognizes are carriage return (\r), line feed (\n), bell (\007), and backspace (\b), so don't expect it to expand tabs. Also, \n is not interpreted as a "newline," but as a line feed. That is, it will get you to the current column on the next line. Use \r to get to the left edge of the current line.

wstr(s, move)
char *s;
int move;
This routine prints an entire string to the screen. If move is true, the cursor ends up positioned just past the end of the printed string. Otherwise, the cursor remains over the first character of the string. Wstr( ) uses wchar( ) as its output function.

An Fgets( ) that Edits

Listing Five (page 20) is an editing input function. Let me point out at the start that this function is general purpose in nature, even though it's written specifically for an IBM PC. It's a relatively simple matter to modify the routines for any terminal that has an addressable cursor. The primary access routine in this module is

```
char *efgets(buf, maxline, fp)
char *buf;
int maxline;
FILE *fp;
```

This is functionally similar to fgets( ), though there are several major differences. First, a pointer to the end (instead of to the start) of the loaded buffer is returned on success. A NULL is still returned on end of file. Second, line continuation is supported. If a line ends with a backslash (\), the backslash is deleted and the line is concatenated with the next line. The major differences from fgets( ) are apparent when fp is set to stdin. In this case several interactive editing functions are supported:

- LEFT-CURSOR (hit the left cursor key) moves the cursor to the left without erasing anything.
- RIGHT-CURSOR moves the cursor to the right.
- ^LEFT-CURSOR (hold down the CTRL key and hit the left cursor key) positions the cursor at the beginning of the previous word.
- ^RIGHT-CURSOR positions the cursor at the beginning of the next word.
- HOME positions the cursor at home position on the current line. Home is

defined as the position of the cursor when the routine was entered. For example, if the cursor was in column 5 when efgets( ) was called, HOME puts it back in column 5. In fact, in this example there's no way to get the cursor into columns 1–4 from efgets. If a line is continued with a \<CR> combination, the cursor is moved to the HOME position of the next line, not to the left edge of the screen.

• END positions the cursor just after the rightmost character on the line.

• ESC is for an abort. The buffer is cleared, but the characters on the screen are not erased. The routine returns immediately. –1 is returned when an ESC is encountered.

• DEL deletes the character on which the cursor is resting and closes up the rest of the line to fill the gap. (i.e., if you delete the X in the string "aaaaXoooo," you end up with "aaaaoooo" on the screen, rather than "aaaa oooo").

• ^H is a destructive backspace. It moves the cursor left one space and then deletes that character, closing up the line to fill the gap.

• ^X erases the entire line and clears the buffer. However, efgets( ) doesn't return.

• ^M (CR) or Enter (LF) positions the cursor at the left edge (not the home position) of the next line and cause efgets( ) to return to the caller. A pointer to the end of the buffer (to the \0) is returned on success, NULL on end of file.

Typing any printing character causes that character to be printed at the current cursor position and the cursor to move right one space. Typing anything else has no effect. The cursor is never allowed to go past the end of the buffer, as specified by the parameter maxline in the efgets( ) call. However, when in editing mode, the cursor is not allowed to go past the end of the current line, even if maxline is longer (the bell will ring if you try). In this case, you can get to the next line with a \<CR>, but you can't edit anything on the previous line.

Conclusion

So that's the beginning. Next month we'll add some more routines to the pile and incorporate them into a simple MS DOS shell. In the following month we'll add various capabilities to that shell.

[*IBM PC readable versions of the listings for the entire shell and for MS DOS versions of various Unix utilities (e.g., grep, ls) will be made available through* DDJ *in the next couple of months. Watch this column for more details.—ed*]

Notes

[1] An excellent description of the IBM BIOS routines and how to use them is in *The Peter Norton Programmer's Guide to the IBM PC*, by Peter Norton (Bellevue: Microsoft Press: 1985).

DDJ

(Listing begins at right)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 191**.

C Chest (Text begins on page 16)

Listing One

```
/*      NEXT.C:         Skip to the next delimiter seperated object
 *
 */

#define iswhite(c) ( (c) == ' ' || (c) == '\t' || (c) == '\n' )


char    *next( linep, delim, esc )
char    **linep;
{
        /* Linep is the address of a character pointer that points to
         * a string containing a series of delim seperated objects.
         * Next will return a pointer to the first non-white object in
         * *linep, replace the first delimiter it finds with a null, and
         * advance *linep to point past the null (provided that it's not
         * at end of string). 0 is returned when an empty string is passed
         * to next(). White space may be used as a delimiter but
         * in this case white space won't be skipped. A delimiter preceded
         * by "esc" is ignored. Quoted strings are copied verbatim.
         */

        register char   *start, *end ;
        int             inquote = 0;

        if( !**linep )
                return 0;

        start = *linep;

        if( !iswhite(delim) )
                for( ; iswhite(*start) ; start++ )
                        ;

        for( end = start; *end && (*end != delim || inquote) ; end++ )
        {
                if( *end == esc  &&  *(end+1) )
                        end++;

                else if( *end == '"' || *end == '\'' )
                        inquote = ~inquote;
        }

        if( *end )
                *end++ = '\0';

        *linep = end;
        return start;
}
```

End Listing One

Listing Two

```
char    cptolower( dest, src )
char    *dest, *src;
{
        /*      Copy src to dest, mapping all upper case letters to lower
         *      case. src and dest may be the same.
         */

        for(; *src ; src++ )
                *dest++ = ('A' <= *src && *src <= 'Z')
                                        ? *src + ('a'-'A') : *src ;
        *dest = '\0' ;
        return dest  ;
}
```

End Listing Two

Listing Three

```
char    *cpy( dest, src )
char    *dest, *src;
{
        /*      Works like strcpy but returns a pointer to the new end
         *      of string (ie. to the null).
         */

        while( *src )
                *dest++ = *src++ ;

        *dest = 0;
        return dest;
}
```

End Listing Three

(Listing Four begins on next page)

C Chest (Listing continued, text begins on page 16)

Listing Four

```
#include <stdio.h>
#include "/lc/dos.h"

/*   VIDBIOS.C        Various cursor and i/o routine using
 *                    the bios interrupts (see below for greater detail):
 *
 *         Copyright (C) 1985 Allen I. Holub. All rights reserved.
 *
 *   Externally accessable routines:
 *
 *    int    getpage  ()                         Get active video page #
 *    void   cursize  (top,bot)                  Set cursor size
 *    void   scur     (posn, page) short posn;   Set cursor position
 *    short  gcur     (pagenum)                  Get cursor position
 *    void   setcur   (posn) short posn;         Set cur pos on current page
 *    short  getcur   ()                         Get cur pos from cur page
 *    void   wchar    (c)                        write a single character
 *    wstr   wstr     (s, move) char *s;         write a string
 *
 */

#define VIDEO_INT      0x10    /*     Video interrupt                    */

#define CUR_SIZE       0x1     /*     Set cursor size                    */
#define SET_POSN       0x2     /*     Modify cursor posn                 */
#define READ_POSN      0x3     /*     Read current cursor posn           */
#define WRITE_TTY      0xe     /*     Write character & move cursor      */
#define GET_VMODE      0xf     /*     Get current video mode & disp pg   */

/*-------------------------------------------------------------------*/

static union REGS       Regs;

/*-------------------------------------------------------------------*/

int     getpage()
{
        /*      Returns the currently active display page number
         */

        Regs.h.ah = GET_VMODE;
        int86( VIDEO_INT, &Regs, &Regs );

        return (int) Regs.h.bh ;
}

/*-------------------------------------------------------------------*/

cursize( top_line, bot_line )
{
        /*     Scan lines are numberd 0 at the top and 7 at the bottom.
         *     if the two are reversed you'll get a 2 part cursor.
         *     Top_line determines the position of the top scan line
         *     of the cursor, bot_line is the bottom. A normal cursor
         *     can be created with cursize(6,7). Cursize(0,7) will
         *     fill the entire area occupied by a character. Cursize(0,1
         *     will put a line over the character rather than under it.
         */

        Regs.h.ch = top_line ;
        Regs.h.cl = bot_line ;
        Regs.h.ah = CUR_SIZE ;
        int86( VIDEO_INT, &Regs, &Regs );
}

/*-------------------------------------------------------------------*/

scur( posn, pagenum )
short   posn;
{
        /*      Modify current cursor position. The top byte of "posn"
         *      value holds the row, the bottom by the column.
         *      Pagenum is the video page number.
         */

        Regs.x.dx = posn     ;
        Regs.h.bh = pagenum ;
        Regs.h.ah = SET_POSN ;
        int86( VIDEO_INT, &Regs, &Regs );
}

short   gcur( pagenum )
{
        /*      Get current cursor position. The top byte of the return
         *      value holds the row, the bottom by the column.
         *      Pagenum is the video page number.
         */

        Regs.h.bh = pagenum  ;
        Regs.h.ah = READ_POSN ;
        int86( VIDEO_INT, &Regs, &Regs );

        return( Regs.x.dx );
```

```
}

/*      Setcur and getcur work just like scur and gcur except that they
 *      access the current video page.
 */

setcur( posn )
short   posn;
{
        scur( posn, getpage() );
}
short   getcur()
{
        return gcur( getpage() );
}

/*----------------------------------------------------------------------*/

wchar( c )
{
        /*      Write a character to the screen in TTY mode. Only normal
         *      printing characters, BS, BEL, CR and LF are recognized.
         *      The cursor is automatically advanced and lines will wrap.
         */

        Regs.h.bl = 0;          /* Use current color */
        Regs.h.al = c;
        Regs.h.ah = WRITE_TTY ;
        int86( VIDEO_INT, &Regs, &Regs );
}

wstr( str, move_cur )
char    *str;
{
        /*      Write a string to the screen in TTY mode. If move_cur is
         *      true the cursor is left at the end of string. If not
         *      the cursor will be restored to its original position
         *      (before the write).
         */

        register short  posn;

        if( !move_cur )
                posn = getcur();

        while( *str )
        {
                Regs.h.bl = 0 ;
                Regs.h.al = *str++ ;
                Regs.h.ah = WRITE_TTY ;
                int86( VIDEO_INT, &Regs, &Regs );
        }

        if( !move_cur )
                setcur( posn );
}

/*----------------------------------------------------------------------*/
#ifdef DEBUG

main()
{
        cursize( 0, 7 );
        wstr( "The large cursor should be on the 'T'", 0 );
        getchar() ;

        cursize( 6, 7 );
}

#endif
```

End Listing Four

Listing Five

```
#include <stdio.h>

/*      EFGETS.C        An editing version of efgets. Recognizes \<CR>
 *                      line termination and supports editing when input
 *                      is from stdin.
 *
 *       Copyright (C) 1985 Allen I. Holub. All rights reserved.
 *
 *--------------------------------------------------------------------
 *
 * Externally accessable routines:
 *
 * void  ptail   (bp, end, move )       Print string from bp to end. Move
 *                                      cursor to end if move is true.
 * char  *egets  (start, bufsize)       Get a string from stdin w/ editing.
 *                                      get at most bufsize-1 chars.
 * char  *getl   (buf, maxline, fp)     like fgets but returns pointer to
 *                                      end of input string on success.
 * char  *efgets (buf, maxline, fp)     like getl but uses egets for
 *                                      standard input rather than getc.
```

(Continued on next page)

C Chest
(Listing continued, text begins on page 16)

Listing Five

```
 *
 *                  FILE *fp;
 *                  char *buf, *bp, *end;
 *                  int  move, bufsize, maxline;
 */

extern int    getpage ();          /* Sources in vidbios.c                 */
extern void   setcur  ();          /*               "                      */
extern short  getcur  ();          /*               "                      */
extern void   wchar   ();          /*               "                      */
extern int    dos     ();          /* Part of the Lattice standard library*/
extern void   movemem ();          /*               "                      */

/*      Values returned from DOS when cursor keys etc. are hit:
 */

#define _LEFT            75
#define _RIGHT           77
#define _CTL_LEFT        115
#define _CTL_RIGHT       116
#define _INS             82
#define _DEL             83
#define _HOME            71
#define _END             79

/* The above are mapped as follows by getkey
 */

#define LEFT             0x80
#define RIGHT            0x81
#define CTL_LEFT         0x82
#define CTL_RIGHT        0x83
#define INS              0x84
#define DEL              0x85
#define HOME             0x86
#define END              0x87

#define BDOS_IN    8      /* raw (non echo) input function        */

#define CNTL_C  0x03             /*  ^C  */
#define CNTL_Z  0x1a             /*  ^Z  */
#define BEL     0x07             /*  ^G  */
#define ESC     0x1b             /*  ^[  */
#define CAN     0x18             /*  ^X  */


/*----------------------------------------------------------------------*/

static int      getkey()
{
        /*      Return a key from the keyboard. Keys are gotten in raw
         *      input mode and mapped as specified above if necessary.
         */

        register int    c;
        static   int    ateof   = 0;

        if( ateof )
                return EOF;

        if( !(c = bdos(BDOS_IN)) )      /* Special function key */
        {
                switch( bdos(BDOS_IN) )
                {
                case _LEFT:             return( LEFT      );
                case _RIGHT:            return( RIGHT     );
                case _CTL_LEFT:         return( CTL_LEFT  );
                case _CTL_RIGHT:        return( CTL_RIGHT );
                case _INS:              return( INS       );
                case _DEL:              return( DEL       );
                case _HOME:             return( HOME      );
                case _END:              return( END       );
                default:                return( NULL      );
                }
        }
        else if( c == '\r' )                    /* map ENTER key to '\n' */
        {
                return( '\n' );
        }
        else if( c == CNTL_C || c == CNTL_Z )
        {
                ateof = 1;
                return EOF;
        }
        else
                return c;

}

/*----------------------------------------------------------------------*/

ptail( bp, end, move )
```

```
register char   *bp, *end;
{
        /*      Print out all characters between bp and end (inclusive)
         *      without modifying the current cursor position. If move
         *      is zero the cursor will not change position, otherwise
         *      the cursor will be left pointing to the character
         *      referenced by end.
         */

        register short  posn;

        if( !move )
                posn = getcur();

        for( ;  *bp && bp <= end;   wchar(*bp++) )
                ;

        if( !move )
                setcur( posn ); /* put cursor under first character */
        else
                wchar('\b');    /* put cursor under last character  */
}

/*----------------------------------------------------------------------*/

char    *egets( start, bufsize )
char    *start;
{
        /*      Get a string with editing. If bufsize is wider than your
         *      screen, strange things will happen when you try to use
         *      the editing functions. If you access this function
         *      via fgetline() then it will input longer lines in 78
         *      character chunks and \<CR> can be used to extend a line.
         *
         *      ^H or BACKSPACE         Destructive backspace, close up
         *                              remainder of string to fill hole.
         *      LEFT   CURSOR           Non-destructive backspace
         *      RIGHT  CURSOR           Move right one character.
         *      ^LEFT  CURSOR           Left  to previous word or line start
         *      ^RIGHT CURSOR           Right to next word or line end
         *      HOME                    Left edge of line
         *      END                     Right edge of line
         *      CR or LF                Terminate line.
         *      ^X                      Erase entire line but don't return.
         *      ESC                     Return a null string immediatly.
         *      DEL                     Delete a current cursor position and
         *                                 close up to fill hole.
         *      Any printing character  Enter that char at cursor posn
         *      Anything else           Ring the bell.
         *
         *      The bell will also ring if you try to move the cursor
         *      past either the left or right edges of the buffer.
         *      Return a pointer to the end of string normally, return 0
         *      on EOF and return -1 when ESC is encountered.
         */

        register char   *bp;    /* Points at current cursor position    */
        register char   *end;   /* Points at largest possible cur pos.  */
        register char   *maxbp; /* Points at rightmost char on line     */
        register int    c ;     /* Current character.                   */
        short           home;   /* place to remember the leftmost cursor
                                 * position.
                                 */

        end     = start + (bufsize-1) ;
        *end--  = '\0' ;
        bp      = start;
        maxbp   = start;

        /*      Fill the entire buffer with spaces
         */

        home = getcur();                        /* Get the current cursor  */
                                                /* position.               */

        if( *bp )                               /* If the buffer isnt empty */
        {                                       /* print out its contents   */
                while( *bp )                    /* and set maxbp to point   */
                    wchar( *bp++ );             /* at the previous end of   */

                maxbp = bp;                     /* string.                  */
                setcur( home );
        }

        for( ; bp <= end ; *bp++ = ' ' )        /* and then fill the rest  */
                ;                               /* of it with spaces.      */

        bp = start;

        /*      bp points into the bufer at the current cursor location.
         *      end points at the righmost place that the cursor movement
         *      commands can get us. There is actually one more place
         *      in the buffer. Get the line:
         */

        while(  (c = getkey()) != '\n'  &&  c != EOF )
        {
                switch( c )
```

(Continued on next page)

C Chest
(Listing continued, text begins on page 16)

Listing Five

```
        {
        case LEFT:              /* Non-destructive backspace      */

                if( bp > start )
                {
                        wchar( '\b' );
                        --bp;
                        break;
                }

                wchar( BEL );
                break;

        case '\b':              /* Destructive backspace          */

                if( bp <= start )
                {
                        wchar( BEL );
                        break;
                }

                wchar( '\b' );
                --bp;

                /* fall through to delete case */


        case DEL:           /* Delete character and close up    */

                if( bp >= maxbp )      /* nothing to delete     */
                        break;

                movmem( bp+1, bp, maxbp-bp );
                *maxbp = ' ';
                ptail( bp, maxbp-- , 0);
                break;

        case CTL_LEFT:      /* Cursor to start of previous word */

                if( bp > start )
                {
                        do {
                                --bp;
                                wchar( '\b' );

                        } while( bp > start && *bp == ' ' );

                        while(   bp > start && *bp != ' ' )
                        {
                                --bp;
                                wchar( '\b' );
                        }

                        if( *bp == ' ' )
                                wchar( *bp++ );
                        break;
                }

                wchar( BEL );
                break;

        case HOME:              /* Cursor to left extreme         */
                bp = start ;
                setcur( home );
                break;

        case ESC:               /* Erase entire line and return */
                *start = 0;
                wchar( '\r' );
                wchar( '\n' );
                return -1 ;

        case CAN:               /* Erase entire line              */
                setcur( home );
                for( bp = start; bp < maxbp; *bp++ = ' ')
                        wchar(' ');

                setcur( home );
                maxbp = bp = start;
                break;

        case RIGHT:             /* Cursor right one character     */

                wchar( ( bp < end ) ? *bp++ : BEL );
                break;

        case CTL_RIGHT:         /* Advance to next word           */

                while( bp < maxbp  &&  *bp != ' ' )
                        wchar( *bp++ );

                while(  bp < maxbp  &&  *bp == ' ' )
```

```
                                        wchar( *bp++ );

                                break;

                        case END:               /* Go to right extremity  */

                                if( bp < maxbp )
                                {
                                        ptail( bp, maxbp, 1 );
                                        bp = maxbp ;
                                }
                                break;

                        default:
                                /*      If we aren't at the right-most extreme
                                 *      of the buffer, move the tail over to
                                 *      make room for the current character,
                                 *      else just print it.
                                 */

                                if( bp <= end  &&  (' ' <= c  &&  c < 0x7f) )
                                {
                                        if( bp < maxbp )
                                        {
                                                if( maxbp < end )
                                                        maxbp++ ;

                                                movmem( bp, bp+1, end-bp );
                                                ptail ( bp, maxbp, 0 );
                                        }

                                        wchar( c );

                                        if( bp < end )
                                                *bp++ = c;
                                        else
                                        {                       /* we're at the  */
                                                wchar( '\b' );/* right margin  */
                                                wchar( BEL ); /* back up and   */
                                                *bp = c ;     /* ring the bell */
                                        }

                                        if( bp > maxbp )
                                                maxbp = bp;

                                        break;
                                }

                                break;
                        }
        }

        /*      Delete trailing whitespace, terminate the string, go
         *      to the next line, and return EOF if we're at end of
         *      file, the end pointer otherwise.
         */

        for( ; *end == ' ' && end >= start ; --end )
                ;

        *++end = '\0';
        wchar( '\r' );
        wchar( '\n' );

        return ( c == EOF  &&  start == end ) ? NULL : end ;
}

/*----------------------------------------------------------------------*/

char    *getl( buf, maxline, fp )
char    *buf;
FILE    *fp;
{
        /*      Works exactly like fgets but returns a pointer to the
         *      end of the string on success.
         */

        register int    c;
        register char   *bp = buf;

        while( (c = fgetc(fp)) != EOF  &&  c != '\n'  && --maxline > 0 )
                *bp++ = c;

        *bp = '\0' ;

        return( (c == EOF && bp == buf) ? NULL : bp );
}

/*----------------------------------------------------------------------*/

char    *efgets( buf, maxline, fp )
char    *buf ;
FILE    *fp  ;
{
        /*    An editing version of fgets.
         *    Works like fgets but recognizes a back-slash at end of line
         *    if fp is stdin then raw i/o is used and various editing
         *    functions are enabled (see egets for details). A pointer
```

(Continued on next page)

C Chest **(Listing continued, text begins on page 16)**

Listing Five

```
         *    to the end of the input string is returned on success or
         *    NULL on end of file.
         */

        register char   *bp, *start = buf ;
        int             linelen, col;

        if( fp == stdin )
        {
                /*  linelen is the amount of space left on the
                 *  current input line. Col is the column componant
                 *  of the current cursor position.
                 */

                linelen = 80 - (col = getcur() & 0xff);
        }
        while(1)
        {
                bp = (fp == stdin) ? egets( buf, min(linelen, maxline) )
                                   :  getl( buf, maxline, fp )
                                   ;

                /*      If egets() found and ESC (bp == -1) or we've
                 *      hit end of file (!bp) or we've seen a
                 *      blank line (bp == buf) the last character on the
                 *      line isn't a \, break; Note that in the first
                 *      case we erase then entire buffer.
                 */

                if( bp == -1 )
                {
                        *(bp = start) = '\0';
                        break;
                }

                else if( !bp  ||  bp <= buf  ||  *(bp-1) != '\\' )
                        break;

                /*      Adjust maxline to compensate for the characters
                 *      already gotten and decrement bp so that we'll
                 *      overwrite the \ on the next pass. Then, if
                 *      we're getting input from stdin, position the
                 *      cursor in its original column but on the current
                 *      line.
                 */

                maxline -= (--bp - buf);
                buf      = bp ;

                if( fp == stdin )
                        setcur( (getcur() & ~0xff) | col );
        }

        return( bp );
}

/*----------------------------------------------------------------------*/

#ifdef DEBUG

main()
{
        static char buf[80] ;

        printf("              1         2         3         4\n");
        printf("     1234567890123456789012345678901234567890\n");
        printf("     ");

        while( efgets(buf, 40, stdin)  >  0 )
        {
                printf("     %s<---\n\n", buf );
                printf("              1         2         3         4\n");
                printf("     1234567890123456789012345678901234567890\n");
                printf("     ");
                *buf = 0;
        }
}

#endif
```

End Listings

Windowing Operating Environments

TopView, GEM, and Windows

by Michael Swaine

What's really going on inside GEM, beneath TopView, behind Windows?

It's a good thing Bill Gates isn't thin-skinned. Otherwise he might take offense at the way so many programmers are spending so much time enhancing, augmenting, bypassing, subverting, taming, masking, and hiding MS DOS. And now Microsoft is getting into the act with Windows. This article examines Windows and two other "windowing, multitasking programming environments" for MS or PC DOS machines: IBM's TopView and DRI's GEM.

You could argue that the seeds were planted by no less a gardener than the Jolly Blue Giant when IBM released its PC with no operating system but with three operating system options: PC (a.k.a. MS) DOS; CP/M-86; and SofTech's UCSD p-System. Although IBM's subsequent pricing strategy strongly hinted that PC DOS was the PC's DOS, the other operating systems never entirely took the hint and went away. Nor did they even get humble: Consider NCI's recent ads for its p-System that announce a new release for the whole PC family (except for jr, whom the family never mentions). The ads, directed at programmers, suggest that with the p-System you can have compatibility with the operating system that your customers, in their ignorance, want (DOS), but can still work in a good programming environment (not DOS).

Although there were precursors, DRI should probably get the credit for bringing concurrent processing to personal computers with its concurrent version of CP/M. It soon became clear to DRI that to sell concurrency on IBM PCs it had to develop a concurrent PC DOS, acknowledging Microsoft's control of the PC operating system area. But DRI had raised the issue of concurrency on a single-user computer.

After DRI opened the concurrency window, there came the pop-ups, an undisciplined horde of dwarf programs that demonstrated even to ordinary mortals the benefits of some approximation of concurrency. They presented object lessons in the hazards of operating system anarchy as well. It was cute when Eastman Kodak put a package under the tree labeled "Open me first"; it was less appealing to confront a dozen software packages all clamoring "Load me first."

What allowed pop-ups to show off their concurrency was windows, an architectural feature opened by Xerox and polished by Apple. The success of the concept of windows is exemplified by its employment in dWindow, a non-operating system application. For years, Ashton-Tate's dBASE II set the standard for austerity in user interfaces—you can't get much simpler than a single period prompt. Liberty-Bell Software's dWindow does a dazzling cathedral window treatment on dBASE II and III that makes them look like entirely different products. But it is at the operating system level that the fenestration of the interface has become obligatory, and it was Apple's Macintosh that made it so.

Bill Gates and his Microsoft programmers were among the first developers to peek behind the blinds at Apple, and they were not slow to announce Microsoft's own window-oriented product, which they called Windows. Sometime between the announcement of Windows and its release, the Korean War ended, Alaska and Hawaii attained state-

hood, and Digital Research programmers conceived and brought out a windowing product called GEM, whose visual display, Apple recently noticed, looks rather like the Mac's. Meanwhile, IBM caught the drift, fumbled through its pockets, and pulled out a windowing, multitasking operating environment that it christened TopView.

With at least three windowing, multitasking operating environments to load on top of, stack next to, or wrap around DOS, the archetypal "fully loaded PC" was beginning to look like a pop-up on steroids. In fact, or at least in *PC Week*, Peter Norton described a dream he had in which he tried to run TopView, GEM, and Windows simultaneously. His hard disk flew to Poland.

Smart Money Talks a Lot

A programmer must face more serious questions of compatibility. How does each product work with DOS? How will existing and future programs run in each of these environments? How well will they run? What are the costs of laying another sheet of software between programmer and machine? We asked such questions of development-team members for each of the three products. But as the answers came back, industry analyses were circulating that questioned the viability of at least two of the products (the two that had actually been released). Were we wasting our time asking questions about these products?

Although some developers are producing TopView-compatible applications, the product has clearly not captured the imagination of users or developers. The recent agreement between IBM and Microsoft has fueled much speculation about the viability of TopView, and many observers seem to be betting on Microsoft. One popular argument is that IBM will create two systems: one using Windows and Microsoft's DOS, the other using a DOS-free descendant or unrelated successor to TopView—in any case, a proprietary operating system. Some, including IBM watcher Andy Seybold, speculate that the IBM-Microsoft agreement may have little to do with IBM's long-term goals and that IBM will use TopView as the centerpiece of its future operating systems. At least one analyst insists that IBM's history demonstrates that it will eventually have a proprietary operating system on the PC. But IBM has broken tradition several times in the short history of the PC, and the venerable wisdom about what IBM always does may not be as wise now as once it was. It does seem that it is getting harder to second-guess IBM, and that the benefits from succeeding therein may be even more dubious today than in mainframe days.

The smart money is writing off GEM in the IBM-compatible market, reasoning that if IBM owns the hardware and Microsoft owns DOS, where does that leave a company that tries to compete with them on their turf?

Perhaps DRI's success with GEM will have to come in the other end of the dumbbell. Lee Felsenstein, in proselytizing for his Hacker's Mac project, describes the personal computer market as an asymmetric dumbbell with one globe of IBM-compatibles and a smaller globe of Macalikes. He argues that the smaller globe can grow and prosper only with compatibility and proposes a radical strategy for forcing compatibility on unwilling Apple, Commodore, and Atari. Such a development would likely benefit DRI and GEM, but is it likely?

Perhaps not, given the fact that Apple has pressured DRI into changing GEM to decrease its similarity to the Macintosh's visual interface. GEM as originally released had to be terminated by November 15, that is, DRI had to stop supporting and advertising it. The new version will look less Macintoshish. Apple is also talking with Microsoft about Windows, but in softer tones.

Smart money and dream-machine designers aside, GEM provides MS DOS users and programmers with capabilities that Windows and TopView lack. But TopView and Windows have their own distinct, desirable features.

Three facts argue against accepting the judgment of the smart money too hastily. First, the three operating environments offer three different sets of capabilities to users and software developers. Second, no one needs any of these products; they are all frosting on the DOS, and users may decide among them—or against all of them—on grounds that smart money would consider dumb. Third, as Bob Frankston pointed out in *InfoWorld*, writing for TopView (or Windows or GEM) limits your market. In any case, we assume there is merit in understanding these products from a programmer's point of view.

Inside GEM

At one level, getting started developing an application that is compatible with GEM is simple. You get the Programmer's Toolkit and start writing. In terms of hardware, you need a PC with half a megabyte of memory and a color-graphics adapter. You should also have a hard disk and a mouse, although you can do development work without them.

Beyond these elementary requirements, one comes up against the fact that GEM is a message-passing program. Years of single-thread procedural programming experience will not prepare you adequately for the different programming model that GEM employs. According to one GEM programmer, the learning curve within DRI during GEM development was two months. Programmers who had never worked with any windowing system took two months to get up to speed in the GEM programming environment. Programmers who had had some windowing experience learned faster but had to unlearn some details that did not transfer.

GEM itself does not communicate directly with DOS; that is, the Virtual Device Interface and Window Services don't. The GEM Desktop does. File manipulation is handled through DOS calls. GEM supports a variety of devices and is expected to support the AST/Ashton-Tate/DRI/Quadram expanded-memory specification, which will become particularly significant when the multitasking version of GEM is released.

GEM compares more directly with Windows than with TopView; TopView is character-oriented and truly multitasking and GEM and Windows are neither. In comparing GEM and Windows as programming environments, Windows developers point to powerful features, and GEM de-

velopers talk about a clean programmer's interface. But the clearest advantages GEM has over Windows at the moment seem to be that GEM has been out long enough to land significant committed and producing OEMs, including Atari and Apricot, and the leverage that this gives the developer in porting an application to different environments.

What does the Apple-DRI settlement mean to programmers who have developed GEM-compatible software or who are thinking about doing so? Perhaps not much. The changes in GEM forced by Apple appear to be essentially cosmetic, and though the impact on the GEM Desktop will be significant, the programming impact may be minimal.

Beneath TopView

The chief difference between TopView on the one hand and GEM and Windows on the other is that TopView is truly multitasking. You can see multitasking in action if you bring up a visually active BASICA program in two TopView windows simultaneously. You'll see the program doing its thing in parallel with itself.

Because it is a multitasking operating environment, TopView permits the development of multitasking applications. The application developer can produce a task with its own subtasks. Then, in addition to running the application in multitasking mode with other tasks, TopView will multitask the application's subtasks. Tasks can communicate; if you get the object handle for a task you can send it a message.

Memory management under TopView is as simple as it is under DOS. When a program begins, it is assigned memory according to the demands of its fixed Program Information File. Henceforth it can get no more nor less. On a program's termination, its memory is freed.

TopView does not replace DOS. It sits atop DOS, handing off file I/O and other system functions to the system. It is at the character I/O level that TopView butts in, redirecting all character I/O. Among other things, this means that you can freely mix TopView and DOS calls. You can put the user in DOS under TopView and have the fact that TopView is active be unknown to them. (A Getversion call will show the user that TopView is active.)

Although TopView is character-oriented rather than graphics-oriented, it will work with a standard monochrome or color screen and anything up to the EGA. TopView will work with the EGA but won't take advantage of its extended graphics capabilities.

TopView developers are trying to woo other developers, arguing that you can bring a more powerful application to market more quickly if you write for the TopView environment—you can take advantage of a standardized display style, a toolkit of window design aids, and other development tools. You can also use what TopView designers call full-screen input. This wooing of developers has not been entirely unsuccessful; the Trio micro-to-mainframe product and the Lattice Topview Toolbasket are significant TopView-compatible products.

For existing programs, you can make use of as much of the windowing capability as makes sense. For some applications this may be useless, but the capability is there. The implementation of DOS services shows what IBM projects for future applications—making logical use of several windows rather than just throwing a frame around a full-screen display.

Behind Windows

Because Microsoft has good sources on what future versions of DOS will require, its programmers can do what they admonish independent developers not to do—work around DOS. Windows runs "side by side with DOS." When Windows comes up, its DOS executive replaces command.com, which is no longer needed.

Windows is made up of three pieces: two are the external pieces that everyone sees, the user interface and the graphics device interface (GDI). The third is the kernel, which interfaces to whatever kind of MS DOS is on the machine. This interface changes considerably with different versions of DOS and hides the operating environment from Windows.

The user interface differs from the Mac/GEM/TopView approach in tiling the screen with windows that don't overlap other windows.

At the GDI level is an interrogating interface: if a device says that it can only do bitblt, the interface will simulate everything else in the software using bitblt; if the device says it can do complex polygons with hash filling, GDI hands off to the device. The device manufacturer fills in capabilities up to some level and Windows simulates the rest. The goal at the GDI level was device independence—the ability to change one line of code and have output go to a different device.

At the level of the kernel, Windows has true compacting global memory management; it does not have true preemptive multitasking. It can allocate, reallocate, dynamically free and restore data. Its task handling is round robin nonpreemptive multitasking—the application must yield control. There are mechanisms for implicit yielding when the application is waiting for something.

The reason Microsoft gives for not implementing true preemptive multitasking is suggestive: DOS is not reentrant. If you preempt a task and it's in DOS at the time, you'd best get back to DOS quickly. So a good chunk of the time slicing would not be beneficial. That, Windows developers claim, is why TopView is slow. Rick Dill of the Windows design team says, "We think multitasking really belongs at the operating system level, and it will get there eventually." He means in MS DOS.

Windows started from the Smalltalk push model and changed to a more procedural approach only when Microsoft found that programmers didn't work well with its implementation of the push model. The general structure of a Windows application is: initialization; Windows-required initializations (because Windows does not require that an application be installed it does require that the application register itself with Windows when it starts); creating a window; and the main program while loop—Getmessage, Translatemessage, Dispatchmessage.

One of the hidden features is the ability for applications

written by different vendors to be integrated via cut and paste as well as the via the messaging mechanism. For example, Multiplan could register a message that says "cell changed." A presentation-graphics program could know that that message means it can look in the clipboard for the contents of the cell and automatically update the chart being displayed.

Windows is 85 percent written in C; the low-level part is written in assembly. Windows supports Pascal, FORTRAN, C, or Microsoft's macro assembler. Because of the new .exe format, programmers will have to use Microsoft's new linker.

According to Dill, Microsoft hopes to entice software developers with the knowledge that, in conforming with Windows, they're building a future DOS-conforming application. Windows is a sneak peek into DOS. (On the other hand, Dill admits there are software developers who have disassembled the .exe file and know more about the innards of DOS than he does.)

But does all this just get in the software developer's way? Most serious software developers find that they must work around DOS to produce a competitive product. Won't software developers find Windows just as encumbering?

"Our bitblt," Dill responds, "is faster than yours." He's serious: Microsoft's bitblt is a general-purpose source-pattern-destination function supporting four operators. The result is that there are 256 different operations available: others implement some subset, often 16, of those. The Windows graphics routines can move a graphics block around the screen almost as fast as the data can be moved through memory. Whether that answers the question is moot.

One significant change Microsoft is introducing is a new .exe file format. Future versions of DOS (of which Windows is a hint) will need more information in the .exe file. To maintain some compatibility, Microsoft has grafted the old .exe header, code, and data on top of the new .exe header. The old tells where the new begins, and the new has its own code and data. It also has something totally new called resources. Resources (like menus and dialogue boxes) can be changed without messing with the .exe file. Thus, one can have one binary that works worldwide, with the resources supplying the language-specific information.

What, then, of the .com files? In the next version of DOS, .com files go away. DOS 3 is the last Microsoft operating system in which memory-image programs will be supported under the operating system itself. "We just need the information," Dill explains, "to be able to do things like running that application up in high memory and being able to segment it correctly and deal with running an old style 8088 unprotected application under a 286." DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 192**.

BANKSWAP

A Banked Memory Debugging Tool for CP/M Plus

by Albert S. Woodhull

By making extended memory useful to 8-bit processors, CP/M Plus can help to keep 8080-, 8085-, and Z80-based microcomputers in competition with newer 16-bit systems. Although the 8-bit processors can directly address only 64K of memory, under CP/M Plus a megabyte can be efficiently used.

For the programmer there is a catch. Transient programs must reside in a single 64K bank. They have indirect access to alternate banks through the operating system. A user running purchased software will not be bothered by this. Developers and others who need to write or modify code that uses the alternate banks have a serious problem, however. CP/M Plus comes with development tools, such as the RMAC assembler and LINK, that enable one to write code for use in any bank of memory, but the standard CP/M Plus debugging tool, SID, is an ordinary transient program, incapable of examining or changing the alternate banks. The programmer needs new tools or a way to extend the power of the old ones.

My response to this need was BANKSWAP, an extension for SID or DDT that provides access to the alternate banks. I will describe this tool after a brief discussion of how CP/M Plus uses banked memory.

Banked Memory in CP/M Plus

Additional banks of memory can improve the speed and memory availability for a user program even though the program resides entirely in the main bank. The speed increase comes from using the extra memory to buffer disk data and disk directories. The space increase comes from putting part of the operating system in an alternate memory bank, making almost all of the 64K of memory the processor can address available to the user's program. For instance, on an Apple with the Advanced Logic Systems CP/M Card the transient program area (TPA) is 60K, even though the CP/M Plus system itself (exclusive of the CCP) takes up 17K.

User programs cannot directly access data or executable code in alternate banks. The point can be argued, but I believe Digital Research made a wise choice in imposing these restrictions. Most programs that use a lot of memory use it to hold data, and disk buffering can provide an improvement in data access comparable to that of extended memory. Given the inherent limitations of 8-bit processors, it is not clear that allowing user programs to execute code resident in extended memory would be much more efficient than using code overlays swapped in from disk buffers.

Under CP/M Plus, programs must be written as if the extra memory is not there at all. As long as the minimum amount of memory needed for an application is present in the TPA bank, a program will be able to run on any CP/M system. This continues the CP/M tradition of providing portability for programs by using the operating system to make details of hardware irrelevant to user programs.

Apple's Filer is fine until you want to copy files between volumes without losing a BASIC program in memory.

Albert S. Woodhull, Hampshire College, Amherst, MA 01002

The BANKSWAP Program

BANKSWAP is not a stand-alone program; it is an enhancement to SID or DDT that provides additional commands to copy blocks of memory from bank to bank. The normal functions of the debugger can be used on a copy of memory from another bank that has been brought to the TPA bank.

BANKSWAP is relocatable and is not necessarily loaded to the same location each time it is used. For ease of use, BANKSWAP installs a vector to its own entry point at the RST 5 location (28H) during installation. Typing G28 from the SID or DDT prompt brings up the BANKSWAP menu. The initialization process also displays a message to remind the user of the presence of BANKSWAP and the command to access it.

The BANKSWAP menu allows the user to choose the direction of the move, the memory addresses for the source and destination, and the length of the block to be moved. The menu also provides for easy return to DDT or SID and for the eventual removal of BANKSWAP. Copying is done in two steps through a buffer that is also in common memory. I chose to use a relatively small buffer and repeat the process several times in order to make the best use of memory space.

Listing One (page 38) for BANKSWAP.ASM contains comments that explain the operation of BANKSWAP, but I will emphasize a few points I found important in working with banked memory. Although I wrote this program for use on an Apple with the Advanced Logic Systems CP/M Card, there should be no problems in making BANKSWAP work on other implementations of CP/M Plus. The most critical point is to be sure that control is not lost while bank 1 is deselected. This means ensuring that BANKSWAP itself, the stack, and all data areas used are located in common memory. It is possible for a CP/M Plus system to be constructed so that interrupts and system calls can be handled while alternate banks are selected. With insurance in mind, I thought it best to disable interrupts and avoid calls to the standard BDOS entry point while bank 1 is deselected because the vectors on page 0 of bank 1 are then inaccessible.

I used a direct call to a BIOS routine to select the bank of memory. There are several points to mention in regard to that. First, the CP/M Plus documentation is emphatic in stating that direct BIOS calls must not be made by application programs; the reason for this is that under CP/M Plus some BIOS routines are always called from the bank 0 portion of the BDOS and will not return to a program located in bank 1. In fact, a separate BDOS function is provided for gaining direct access to the BIOS. The catch is that this BDOS function prohibits access to one BIOS routine—you guessed it—the one we need for BANKSWAP. Digital Research really doesn't want user programs to try to access other memory banks.

Having decided to take the law into your own hands and call the BIOS bank-select routine directly, you can't do it by accessing the BIOS vector in low memory just any old time for the same reason you can't use BDOS routines any old time—some of the time the low memory the program sees will be in bank 0. For

this reason, I made the initialization portion of BANKSWAP fetch the BIOS vector so the necessary entry point could be calculated and stored locally. I did this only for the BIOS SELMEM routine; the program can be simplified if a BIOS XMOVE routine has also been implemented (this is not provided in the Advanced Logic Systems release of CP/M Plus).

Resident System Extensions

BANKSWAP must run in common memory. This is most easily accomplished by creating a relocatable file to be loaded to the top of the TPA. CP/M Plus has another feature, the Resident System Extension (RSX), to simplify this task.

An RSX is a page-relocatable program segment that CP/M Plus loads to the highest available address, just as DDT or SID are handled under older versions of CP/M. Each RSX module has a prefix, as shown in Listing One. The prefix includes a jump to the BDOS entry point in high memory (for the first RSX installed) or to the last previously installed RSX. Whenever an RSX is installed, the JMP instruction at location 5 is modified to point to it, so a chain of JMPs is traversed whenever a BDOS call is made. The address portion of the JMP at location 5 is also used by programs to detect the decreased size of the TPA, protecting the RSX from being overwritten.

Because all BDOS calls pass through the RSX JMP chain, RSXs can be written to intercept BDOS calls to customize BDOS performance. BDOS interception is not necessary, however. BANKSWAP intercepts only the very first BDOS call after it is loaded—this is a convenient way to force execution of the BANKSWAP initialization phase. The initialization code saves and restores whatever information is being passed to the BDOS and modifies the JMP chain to prevent reinitialization.

An RSX must be connected to a normal .COM program to be installed. CP/M Plus provides a utility, called GENCOM, to do this and also provides the RMAC and LINK programs needed to produce the relocatable program and its relocation bit map. The process is considerably more complicated than assembling and loading a program under CP/M 2, but it is easy to use the CP/M SUBMIT program to direct the process. An RSX can be attached to any .COM file. I have attached BANKSWAP to DDT.COM and SID.COM and it works with both. Listing Two (page 52) shows the RSXMAKER.SUB file that can create BANKSWAP.RSX and connect it to SID.COM.

When BANKSWAP is to be attached to SID or DDT it is assembled with a REMOVE flag set in the RSX prefix. This ensures that when a warm boot occurs upon exiting from SID the space occupied by the RSX will be freed. BANKSWAP can also be assembled to be loaded independently of SID by setting the ALONE equate true. In this case it is still necessary to attach the RSX code to a .COM file, which could be nothing more than a jump to location zero. As the Listing One shows, setting ALONE true resets the REMOVE flag in the RSX header, and it also adds an option to the BANKSWAP menu to allow later removal.

There is a potential problem with the version of BANKSWAP shown in Listing One, but I'm leaving it for someone else to fix. As noted above, BANKSWAP uses the RSX technique in order to be located at the highest available memory address. The actual amount of high memory that is common depends upon the hardware used, and I didn't figure out a way for a program to determine this. If too many RSXs are installed, the top of available memory can be below the common region. In this case BANKSWAP will probably cause a crash. On my Rev. A CP/M card the common-memory limit is at 8000H, which leaves room for a lot of RSXs, so I have never given the problem a high priority.

Using BANKSWAP

Once BANKSWAP has been attached to a copy of SID.COM it will be installed whenever SID is run. Listing Three (page 52) shows part of a typical session with enhanced SID. The initialization process tells the operator that BANKSWAP is available. A G28 command enters BANKSWAP, which displays a menu. I usually bring part of an alternate bank over to bank 1 first, then return to the debugger to examine, disassemble, or alter the copy. In fact, if executable code from some portion of bank 0 is brought over to the corresponding address range in bank 1 the debugger can be used to trace through it—providing that the code doesn't switch banks or access I/O or storage addresses in an alternate bank.

In the Apple environment there is one limitation: an Apple's I/O is all memory mapped in bank 0. If an attempt is made to access bank 0 in the range 6000H to 67FFH it may crash the system because some addresses in this range activate switches on the Apple main board or peripheral cards, including the CP/M Card itself.

Conclusion

One of my first reactions to CP/M Plus was a helpless feeling. I had written my own BIOS for CP/M 2.2 on my S-100 system, and I was accustomed to being able to explore how the system worked on any CP/M machine. With CP/M Plus, portions of the operating system were totally inaccessible to the tools I had on hand.

Most users don't need the kind of access I wanted. However, the initial version of CP/M Plus I received did not support my printer interface properly. I also had proposed to develop CP/M Plus driver software for companies that manufactured large-format disks and RAM-disk add-ons for Apples. The usability of my system as well as potential income depended upon my ability to patch various devices into CP/M Plus.

BANKSWAP solved my problems. It gave me a way to satisfy my curiosity about how the Advanced Logic Systems CP/M Plus BIOS worked, and I was able to figure out how to patch some of my peripheral drivers into CP/M Plus. Finally, I now had the tool I needed to begin to develop and debug enhancements to the supplied BIOS. In the process of developing BANKSWAP I also learned how to create and use RSXs, which are very useful CP/M Plus features.

DDJ

(Listing begins on next page)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 193.**

BANKSWAP (Text begins on page 34)

Listing One

```
                 ; BANKSWAP.ASM
                 ;
                 ; A. S. Woodhull         28 June 83
                 ; rev    1 July 85       -- minor editing
                 ;       20 Oct 83
                 ;
                 ; This is designed to be run as a subprogram under DDT
                 ; or SID, in a banked version of CP/M 3.Ø  The function
                 ; of BANKSWAP is to copy blocks of memory from other
                 ; banks to and from bank 1, using a buffer in the common
                 ; area.  Normal DDT or SID functions can then be
                 ; performed, using the copy of the code in bank 1.
                 ; (Of course, you cannot trace through program segments
                 ; that switch banks or access memory mapped I/O in
                 ; another bank.)
                 ;
                 ; BANKSWAP must reside in the common area of memory, and
                 ; a buffer area through which data can be copied must
                 ; also be present in the common area.  BANKSWAP is to be
                 ; assembled as a Resident System Extension (RSX), which
                 ; will automatically be relocated to the top of the TPA.
                 ; It is assumed that the common area is large enough to
                 ; allow an RSX to fit--if this is not true BANKSWAP will
                 ; not work in its present form.
                 ;
                 ; Because the location of BANKSWAP in memory is not fixed
                 ; a jump through a fixed location is set up when the BANK-
                 ; SWAP code is installed.  In this version the  RST 5 vector
                 ; at 28H is used, but any convenient location on page zero
                 ; may be used.
                 ;
                 ; Under CP/M 3 direct access of BIOS routines is generally
                 ; to be avoided by user programs, since BIOS routines may
                 ; have expectations about which bank is selected when they
                 ; are called.  We will, however, do bank switching through
                 ; the BIOS selmem routine.  For generality we will use the
                 ; BIOS vector at location 1.
                 ;
FFFF =           true:   equ     Øffffh
ØØØØ =           false:  equ     not true
                 ;
ØØØØ =           alone:  equ     false    ;make false if attached to DDT/SID
                 ;
ØØØ1 =           biosv:  equ     1        ;address of wboot in BIOS found here
ØØ4E =           selmem: equ     4eh      ;offset from wboot
                 ;
Ø1ØØ =           buflen: equ     1ØØh     ;move block size
                 ;
                 ; default parameters
1ØØØ =           movcnt: equ     1ØØØh    ;to move 16 pages (4K) at a time
Ø1ØØ =           bØdft:  equ     1ØØh     ;start of block in bank Ø
Ø1ØØ =           bldft:  equ     1ØØh     ;start of block in bank 1
                 ;

                 ; zero page addresses
ØØØ5 =           bdos:   equ     5        ;bdos entry point
ØØ28 =           rstv:   equ     28h      ;RST 5 used as entry vector
                 ;
                 ; BDOS functions used
ØØØ1 =           conin:  equ     1        ;get a char
ØØØ9 =           printf: equ     9        ;print a string
ØØØA =           rdbuf:  equ     1Ø       ;read a line
                 ;
                 ; This is standard prefix for a Resident System Extension
                 ;  see CP/M 3 Programmer's Guide, 1st ed., sec. 4.4, p.168
                 ;
ØØØØ ØØØØØØØØØØserial: db        Ø,Ø,Ø,Ø,Ø,Ø
ØØØ6 C3A2Ø3      start:  jmp     install ;one-time routine
ØØØ9 C3ØØØØ      next:   jmp     Ø        ;altered at installation
```

(Continued on page 40)

NEW LANGUAGE BREAKS OLD RULES. GIVES PROGRAMMERS POWER, SPEED AND SIMPLICITY.

Try this remarkable language, PROMAL™, for 30 Days AT NO RISK and...

We think you'll be thrilled with this breakthrough system when you discover its power, ease of use, and dazzling performance on your IBM PC, Apple IIe/IIc, or Commodore 64. **But we don't expect you to accept our claims for PROMAL without proof,** so we invite you to explore the power of PROMAL on your own during our 30-day trial period.

Broken Rules

Now that PROMAL 2.0 has broken the rules, a structured language doesn't have to be slow, unwieldy and difficult to use. PROMAL is fast, elegant, and simple.

What Is PROMAL?

PROMAL stands for **PRO**grammer's **M**icro **A**pplication **L**anguage. But PROMAL is more than a high-level language, it's a total structured programming development system with a fast, one-pass compiler, a versatile full-screen editor, plus an integrated machine-language subroutine library. And for APPLE and Commodore systems it includes a DOS-like system "Executive."

Better By Design

PROMAL was designed from "scratch" for **optimum performance** and ease of use on microcomputers. It has a simplified syntax with no awkward terminators like ";" or "}" and **indentation is part of the syntax,** so structuring your code is natural and easy. Just compare PROMAL with BASIC in this example:

Equivalent Program Segments

```
;-------PROMAL--------
REPEAT
   PROMPT_AT 5,24, "Add/Chg/Quit?"
   IF Reply = 'A'
      ADD Item
      New_Items = New_Items + 1
   ELSE IF Reply = 'C'
      CHANGE Item
UNTIL Reply = 'Q'
```

```
11910 REM -----BASIC----------
11920 CL = 5:LN = 24:PR$ = "Add/Chg/Quit?"
11925 GOSUB 9490:REM GET REPLY
11930 IF RP$<>"A" THEN 11950
11940 I9 = IT:GOSUB 10100:REM ADD
11945 NI = NI + 1:GOTO 11920
11950 IF RP$<>"C" THEN 11970
11960 I9 = IT:GOSUB 6050:REM CHG
11970 IF RP$<>"Q" THEN 11920
```

PROMAL is readable and understandable. You see the logic from the structure. And PROMAL lets you call procedures by name–so no more GOSUBs. But there's more.

Slick Editor

Editing your source is a snap with the specially-designed and integrated **full-screen Editor**–it not only helps you structure your program, it **even finds compilation errors**– automatically.

Quick Compiler

The compiler is a **lightning-fast,** one-pass, recursive descent design. On the IBM PC it crunches source to object at **2000 lines per minute,** and it's equally impressive on the Apple and C64. And your PROMAL source code is **portable from machine to machine.** That means your source can be used on all PROMAL target machines.

Run-Time Speed Demon

PROMAL blows away Apple II and C64 languages from BASIC and PASCAL to FORTH. (Send $3 for a copy of our full benchmark report.) It's **2000% faster** than BASIC. And on a normal IBM PC, the native 8088 code from **PROMAL beat Turbo Pascal 3.0** by 10% on the standard sieve benchmark!

DOS For Those Without

If you don't have a real "DOS," then PROMAL gives you a **true operating system** environment with the built-in operating system Executive. (See box.)

PROMAL 2.0 FEATURES

COMPILED LANGUAGE

- Structured indentation syntax
- No line numbers or terminators
- Long variable names (31 characters)
- Global, Local, & Argument variables
- Byte, Word, Integer & Real data types
- Decimal or Hex number types
- Functions & Procedures with passed arguments
- Predefined DATA of any type
- Multi-Dimensional Arrays (any type)
- Strings & pointers
- Control Statements: IF, IF-ELSE, WHILE, FOR, CHOOSE, REPEAT-UNTIL, BREAK, NEXT, INCLUDE, ESCAPE, REFUGE
- Bit-operators, shifts, type casts
- Variables at any memory location
- Simple Machine Language interface
- Recursion supported
- Program chaining and overlays (IMPORT/EXPORT)
- Separate compilation of modules
- Load and run relocatable M/L programs
- Compile errors trapped for Editor

EXECUTIVE (APPLE II & C64 Only)

- Command driven, with line editing
- Multiple user programs in memory at once
- Function key definitions
- Progam abort and pause
- Prior command recall
- I/O Re-direction & batch jobs
- "DOS"-like commands: COPY, RENAME, DELETE, display FILES, TYPE, HELP, etc.
- Memory MAP, SET, and display commands

EDITOR

- Full-screen, cursor driven
- Function key controlled
- Line insert, delete, search
- String search and replace
- Block copy, move, delete & file read/write operations
- Auto indent, undent support

LIBRARY

- 50 Resident Machine-language commands
- Call by name with arguments
- String handling (9 routines)
- Re-directable I/O (STDIN & STDOUT)
- Formatted numeric output
- Decimal & Hexadecimal I/O
- Block fill/move/read/write
- Cursor control & line editing
- Data type conversion
- Random number function
- Real function support (in PROMAL): ABS, ATAN, COS, EXP, LOG, LOG10, POWER, SIN, SQRT, TAN
- Modem device support & much more

Outside Opinion

Naturally we're enthusiastic about PROMAL, **but here's what other programmers are saying:**

"Excellent... an ideal development system.... Well done indeed!"

M. T. V.
Naperville, Ill.

"I am... so amazed by PROMAL... I cannot believe the high degree of excellence of this entire package."

C. P., Ph.D.
Ridgeway, New York

"I don't know that I've ever seen a [system] as thoughtfully designed and as skillfully executed as PROMAL. Its logic and ease of programming are truly remarkable. Its speed of execution is phenomenal... congratulations."

E. C. R.
Alexandria, VA

Safety In Numbers

SMA, Inc. has been satisfying customers (over 100,000) since 1982 with innovative microcomputer products. Now you can join our thousands of satisfied PROMAL users, by trying it today.

Try It For 30 Days On Us

Send us some bucks and we'll send you PROMAL on trial for 30 days. **If for any reason whatsoever you are not satisfied, just send it back** for a quick refund of your purchase price. No questions asked. No risk.

How To Order.

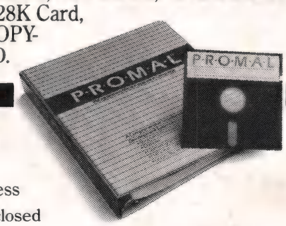
Call TOLL-FREE to order with your credit card or use the handy order form below to send in your check or money-order for your 30-day trial. Don't wait, you deserve the power of PROMAL today!

1-800-762-7874

In NC: 919-878-3600

SMA
Systems Management Associates, Inc.
3325 Executive Drive, Dept. D-2
Raleigh, North Carolina 27609

PROMAL runs on IBM PC/PCjr with 192K, Commodore 64/128, APPLE IIc, or APPLE IIe with 80 Col. 128K Card, and is NOT COPY-PROTECTED.



Order Form for PROMAL 30-Day Trial!

My system is (check one)
☐ IBM PC/100% compatibles ☐ APPLE IIc/IIe
☐ COMMODORE 64/128

Please RUSH me:
☐ PROMAL Developer's System–Compiler, Editor, Library, Demo disk, 280-page manual, (Plus Executive for Apple and C-64) and stand-alone program generation (no royalties).
$99.95 + 5.00 Shipping & Handling
☐ End-User System for Apple IIc/IIe and Commodore 64/128–all features of Developer's Version except stand-alone program generation (Executive needed for program execution)
$49.95 + 5.00 Shipping & Handling
☐ Graphics ToolBox (Apple/C64 only)–20 routines for hi-res graphics: windows, clipping, text-on-graphics using scaled, rotated, user-defined fonts.
$29.95 + 2.50 Shipping & Handling

MEMBER DIRECT MARKETING ASSOCIATION

Please charge my
☐ Visa
☐ MasterCard
☐ American Express
☐ My check is enclosed

Card Number ________
Signature ________ Exp. Date ____
Name ________
Address ________
City ________
State ________ Zip ____

NC residents add 4½% sales tax.
Foreign orders add $20.00 additional shipping.

Listing One

```
000C 0000        prev:    dw      0
                ;
                          if      alone
                 rmvflg:  db      0       ;keep this in memory
                          else
000E FF          rmvflg:  db      0ffh    ;remove when main program ends
                          endif
                 ;
000F 00          nonbnk:  db      0       ;banked system
0010 42414E4B53           db      'BANKSWAP'
0018 00          loader:  db      0
0019 0000                 db      0,0
                 ; *** note: If BANKSWAP is to be attached directly to SID.COM
                 ; or DDT.COM then make rmvflg 0ffh to force removal
                 ;
                 bankexam:
001B 210000               lxi     h,0     ;get stack pointer
001E 39                   dad     sp      ;...and hold it for return
001F 220F04               shld    holdsp  ;...reset SP to a location
0022 310804               lxi     sp,locstk       ;...in common memory
                 ;
                 ; Main loop--exit by Quit or Remove command
                 bankex2:
0025 CD2E00               call    prompt  ;this returns address of sub
0028 CD6D00               call    doit    ;do subroutine addressed by HL
002B C32500               jmp     bankex2
                 ;
002E 111F02      prompt:  lxi     d,menu  ;get ready for menu
0031 3A9103               lda     quietflag
0034 B7                   ora     a       ;suppress menu?
0035 CA3B00               jz      prmpt2
0038 11F402      prmptl:  lxi     d,query ;set for prompt only
003B 0E09        prmpt2:  mvi     c,printf
003D CD0500               call    bdos
                 ; Get a character
0040 0E01                 mvi     c,conin
0042 CD0500               call    bdos
0045 CD7001               call    crlf
                 ; make upper case, reject non-alpha
0048 E65F                 ani     5fh

004A FE41                 cpi     'A'
004C DA3800               jc      prmptl  ;ask again if invalid
004F FE5B                 cpi     'Z'+1
0051 D23800               jnc     prmptl  ;ask again if invalid
                 ; find match in alphtbl
0054 010700               lxi     b,altblen       ;length of table
0057 211102               lxi     h,alphtbl+altblen       ;work back
005A BE          try:     cmp     m
005B CA6300               jz      match
005E 2B                   dcx     h
005F 0D                   dcr     c       ;count down
0060 C25A00               jnz     try
                 ; if c= 0 no match found.  Now form address
0063 211102      match:   lxi     h,addrtbl
0066 09                   dad     b       ;add offset
0067 09                   dad     b       ;again, 2 bytes per table entry
                 ; get the command address
0068 7E                   mov     a,m
0069 23                   inx     h
006A 66                   mov     h,m
006B 6F                   mov     l,a
006C C9                   ret             ;HL has action address
                 ;
006D E9          doit:    pchl            ;call here to use action address
                 ;
                 getbank:
006E 2A9C03               lhld    b0start ;setup addresses
```

(Continued on page 42)

FREE **8087 VERSION with orders placed prior to Dec. 15.**

We've just invented an Algorithm Developer.

It just happens to be the best **programmable software calculator** on the market today.

```
amort.CFG
6-amrtp
months=months+1;pr=((pmt*12/int)/(1+int/12)
^(mths+1))*[1-(1+int/12)];@next
7-amrti
in=-pmt-((pmt*12/int)/(1+int/12)^(mths+1))
*[1-(1+int/12)];@next
8-sumr
prinpd=prinpd+pr;intpd=intpd+in;mths=mths-1;
@ifqt (mths==168);@goto(6)
```

A sample formula listing

PRO/SCI™ 99 bucks and a phone call.

Consider its performance.

- Programmable in simple BASIC—like statements
- Runs in *both* foreground *and* background modes
- Set up: Requires one key stroke
- Runs in background mode with most popular products (e.g. LOTUS, WordStar, dBase)
- Selects and moves rows of numbers *from* foreground document *to* calculator for *any* computation (e.g., standard deviation for 22 rows of numbers requires six key strokes)
- Returns results (at user's option) of all calculations to the foreground document
- Single-Step mode aids in algorithm de-bug
- Spreadsheet comparison
  - — Greater range of formula capabilities
  - — Access requires only key stroke
  - — Transfers numbers to calculator without retyping
- Full formula line editing
- Ten formulas per formula file
- Unlimited number of formula files
- 8087 version available
- All calculations double precision
- Includes Accountant's style calculator with HEX, OCT, and BIN!
- Runs with SIDEKICK — in all modes

Functions

sin, cos, tan, asin, ln, exp, log, sqrt, abs, int, frac, round, mean, std, var, perm, comb, mod, rand, sum, min, max, integrate

Operators

arithmetic: +, −, *, /
logical: #AND#, #OR#, #NOT#
relational: >, <, >=, <=, ==, < >
exponentiation: **, ∧
factorial: !

Programming Statements

Conditional and absolute looping and branching
Indexing by variable increment and test
Formula con-catenation (to 1000 columns)
Printing statements for line printer tabulations
Debug mode — single-step and display of all variables
Displayed decimal point control

Constants

pi, e, c, K, h, q, R, N_0, G, g

Send me **PRO/SCI**™ for $99 and include 8087 Version at NO EXTRA CHARGE.
(Plus $5.00 shipping & handling)

☐ American Express ☐ VISA
☐ Money Order ☐ MasterCard
☐ Check

Card No. ____________
Expiration Date ____________
Signature ____________

Requires IBM PC or true compatible with 128K RAM and one disk drive and MS-DOS 2.1 or higher

NAME ____________
ADDRESS ____________
CITY/STATE/ZIP ____________
TELEPHONE ____________

California residents add 7% sales tax. Outside U.S.A. add $15.00. (If outside of U.S.A. payment must be by bank draft payable in the U.S. and in U.S. dollars.) Sorry, no C.O.D.

SYMSOFT
P.O. Box 4477
Mtn. View, CA 94043
Orders Taken:
Mon.-Fri.; 8am-4pm
800/227-6703
OUTSIDE CALIFORNIA
800/632-7979
INSIDE CALIFORNIA

Circle no. **146** on reader service card.

Listing One

```
0071 220A04             shld    source
0074 2A9E03             lhld    blstart
0077 220C04             shld    dest
007A 3AA103             lda     length+1        ;low byte ignored
007D 320E04             sta     count
                ; page by page take a chunk of bank 0 to buffer, then move
                ; to bank 1 destination
                get2:
0080 F3                 di                      ;be sure no interrupts
0081 3E00               mvi     a,0
0083 2A0804             lhld    selmv
0086 CD6D00             call    doit
0089 CDF400             call    getbuf
008C 3E01               mvi     a,1
008E 2A0804             lhld    selmv   ;point to BIOS selmem routine
0091 CD6D00             call    doit
0094 FB                 ei              ;interrupts safe again
0095 CDFD00             call    putbuf
0098 110001             lxi     d,buflen
009B 2A0A04             lhld    source
009E 19                 dad     d
009F 220A04             shld    source
00A2 2A0C04             lhld    dest
00A5 19                 dad     d
00A6 220C04             shld    dest
00A9 210E04             lxi     h,count ;repeat for required # of pages
00AC 35                 dcr     m
00AD C28000             jnz     get2
00B0 C9                 ret
                ;
                putbank:
00B1 2A9E03             lhld    blstart
00B4 220A04             shld    source
00B7 2A9C03             lhld    b0start
00BA 220C04             shld    dest
00BD 3AA103             lda     length+1        ;low byte ignored
00C0 320E04             sta     count
                ; page by page, move bank 1 data to buffer, then move it
                ; to bank 0
                put2:
00C3 CDF400             call    getbuf
00C6 F3                 di              ;be sure no interrupts
00C7 3E00               mvi     a,0
00C9 2A0804             lhld    selmv
00CC CD6D00             call    doit
00CF CDFD00             call    putbuf
00D2 3E01               mvi     a,1
00D4 2A0804             lhld    selmv
00D7 CD6D00             call    doit
00DA FB                 ei              ;interrupts safe again
00DB 110001             lxi     d,buflen
00DE 2A0A04             lhld    source
00E1 19                 dad     d
00E2 220A04             shld    source
00E5 2A0C04             lhld    dest
00E8 19                 dad     d
00E9 220C04             shld    dest
00EC 210E04             lxi     h,count
00EF 35                 dcr     m
00F0 C2C300             jnz     put2
00F3 C9                 ret
                ; source to buffer
                getbuf:
00F4 2A0A04             lhld    source
00F7 111104             lxi     d,buffer
00FA C30401             jmp     pbl
                ; buffer to dest
                putbuf:
```

(Continued on page 44)

Listing One

```
00FD 2A0C04          lhld    dest
0100 111104          lxi     d,buffer
0103 EB              xchg
               ; common code for getbuf and putbuf
0104 010001    pbl:   lxi     b,buflen
               ; move (BC) bytes from (HL) to (DE) (could use BIOS move
               ; routine for this)
0107 7E        move:  mov     a,m
0108 12               stax    d
0109 23               inx     h
010A 13               inx     d
010B 0B               dcx     b
010C 78               mov     a,b
010D B1               ora     c
010E C20701           jnz     move
0111 C9               ret
               ;
               ; Go back to SID/DDT
0112 116403    quit:  lxi     d,qmsg  ;say how to get back
0115 0E09             mvi     c,printf
0117 CD0500           call    bdos
011A 2A0F04           lhld    holdsp  ;restore stack pointer
011D F9               sphl
011E FF               rst     7       ;back to DDT or SID
               ;
                      if      alone
               ; Set for removal on termination of SID or DDT
               remove: lxi    h,rmvflg         ;set the remove flag
                      mvi     m,0ffh           ;...in the RSX prefix
                      lxi     h,rstv           ;then wipe out the entry
                      mvi     m,0ffh           ;...JMP with an RST 7
                      rst     7                ;leave via the debugger
                      endif   ;alone
               ;
               ; Set up addresses for move, also set up length
011F 111703    adset: lxi     d,b0id  ;tell current bank 0 addr
0122 0E09             mvi     c,printf
0124 CD0500           call    bdos
0127 2A9C03           lhld    b0start ;get the address
012A CDDA01           call    addro   ;and print it
012D 219C03           lxi     h,b0start
0130 CD8101           call    update  ;enter hex to (HL)
0133 112503           lxi     d,blid  ;do it again for bank 1
0136 0E09             mvi     c,printf
0138 CD0500           call    bdos    ;tell
013B 2A9E03           lhld    blstart
013E CDDA01           call    addro
0141 219E03           lxi     h,blstart
0144 CD8101           call    update  ;get new address, if any
               ; Can fall through from adset or enter directly here to set
               ; length of block moved
0147 113303    lnset: lxi     d,lnmsg ;tell current length
014A 0E09             mvi     c,printf
014C CD0500           call    bdos
014F 2AA003           lhld    length
0152 CDDA01           call    addro
0155 21A003           lxi     h,length
0158 CD8101           call    update  ;offer to change it
015B CD7001           call    crlf
015E C9               ret
               ;
               ; Toggle menu off/on
015F 3A9103    xpert: lda     quietflag
0162 2F               cma                      ;toggle
0163 329103           sta     quietflag
0166 C9               ret
               ;
               ; Get here on invalid command
```

(Continued on page 46)

VEDIT PLUS

We have a Surprise for You.

NEW FEATURES MAKE IT BETTER THAN EVER

VEDIT is the most often reviewed and highly acclaimed text editor ever. And no wonder - we have listened to and supported our users for almost six years and have continually improved our products by incorporating hundreds of their suggestions. Now we have a revolutionary new version of VEDIT PLUS with all the program development features you've asked for.

The unique on-line interactive help is a first - it can be easily expanded or edited. You can create your own on-line help with menus and sub-menus for compilers, style guides, even non-computer subjects.

NEW FEATURES

- Memory management supports up to 1 M Byte
- MS-DOS, PCDOS pathname support
- Interactive on-line help
- Paragraph justification
- Create your own edit functions with macros
- Auto-execute macros as application programs
- Easy to use installation/customization program for keyboard assignment, screen attributes, etc.

The supplied macros are menu driven and immediately usable. Sort a mailing list. Perform a full screen file comparison and update one file from the other - merge the work done by two programmers on the same file. Replace the command prompt with a menu for new users. And an exceptional 8080 to 8086 translator is an inexpensive option.

Your own editing macros are easy to write and will automate otherwise tedious chores. You can use a macro the instant you type it or load it from disk - there is no time wasted compiling or linking. Macros are accessible from function keys - allowing you to create your own editing functions.

- Simultaneously edit up to 36 files of unlimited size
- Unlimited file handling - merge, split, extract, view, print and more
- Search with wild cards and patterns
- On-line integer calculator
- Complete macro programming language including:
  IF-THEN-ELSE and looping
  User prompts and input
  Algebraic expressions with variables
  Complete TECO capability
- Free macros

VEDIT PLUS isn't only for programming. The word processing features include word wrap, paragraph formatting with justification, printing and horizontal scrolling for spreadsheets. Converts WordStar files and edits dBASE™ source files. Add V-PRINT for fancier features such as automatic index generation and printer font support (incl. laser printers). Then add V-SPELL, the ultimate spelling corrector, and you'll have a word processing system that can do virtually everything you need.

As a programming or writing professional, VEDIT PLUS gives you refinement, power and flexibility you will never outgrow.

VEDIT PLUS supports virtually every MS-DOS, PCDOS, CP/M-86 and CP/M computer and is a favorite among OEMs. VEDIT PLUS is not copy protected. Order direct or from any of our over 400 dealers. $225. (Discounts available for current VEDIT users.) Site licensing and educational discounts.

CompuView

1955 PAULINE BLVD., ANN ARBOR, MICHIGAN 48103
(313) 996-1299, TELEX - 701821

BANKSWAP (Listing continued, text begins on page 34)

Listing One

```
0167 11FA02     na:     lxi     d,namsg         ;say can't do it
016A 0E09               mvi     c,printf
016C CD0500             call    bdos
016F C9                 ret
                ;
0170 C5         crlf:   push    b
0171 D5                 push    d
0172 E5                 push    h
0173 F5                 push    psw
0174 111403             lxi     d,crlfstring
0177 0E09               mvi     c,printf
0179 CD0500             call    bdos
017C F1                 pop     psw
017D E1                 pop     h
017E D1                 pop     d
017F C1                 pop     b
0180 C9                 ret
                ;
                ; get string, do nothing if null, else convert, store at (HL)
                update:
0181 E5                 push    h               ;save the location
                ; loop back here if input is not valid
0182 210000     updl:   lxi     h,0             ;initial buffer
0185 229303             shld    inbuf+1
0188 114203             lxi     d,chquery       ;say what's up
018B 0E09               mvi     c,printf
018D CD0500             call    bdos
0190 119203             lxi     d,inbuf
0193 0E0A               mvi     c,rdbuf         ;read console until <ret>
0195 CD0500             call    bdos
0198 CD7001             call    crlf
019B 3A9303             lda     inbuf+1         ;get length of hex string
019E B7                 ora     a               ;check for 0 length input
019F C2A401             jnz     convert
                ; null string, go back
01A2 E1                 pop     h               ;retrieve value at entry
01A3 C9                 ret
                ;
                ; Convert the hex string in the buffer to binary
                convert:
01A4 210000             lxi     h,0     ;start with a zero
01A7 47                 mov     b,a     ;hold length in B
01A8 119403             lxi     d,inbuf+2
01AB 1A         conv2:  ldax    d       ;get first (or next) char
01AC 13                 inx     d
01AD FE60               cpi     60h
01AF DAB401             jc      conv3
01B2 E65F               ani     5fh     ;make lower case if necessary
01B4 D630       conv3:  sui     '0'
01B6 FA8201             jm      updl    ;must be valid hex, 0..9, A..F
01B9 FE0A               cpi     0ah
01BB DACA01             jc      num     ;jump if a good numeric
01BE D607               sui     7
01C0 FE0A               cpi     0ah
01C2 DA8201             jc      updl    ;error if not good alpha
01C5 FE10               cpi     10h
01C7 D28201             jnc     updl    ;error if not good alpha
01CA 29         num:    dad     h       ;multiply current val by 16
01CB 29                 dad     h
01CC 29                 dad     h
01CD 29                 dad     h
01CE 85                 add     l       ;add new least significant digit
01CF 6F                 mov     l,a
01D0 05                 dcr     b       ;countdown the digits
01D1 C2AB01             jnz     conv2
01D4 EB                 xchg            ;result to DE
01D5 E1                 pop     h       ;HL at entry says where to it
```

(Continued on page 48)

Listing One

```
01D6 73                 mov     m,e
01D7 23                 inx     h
01D8 72                 mov     m,d
01D9 C9                 ret
               ;
               ; Print HL as hex
               addro:
01DA D5                 push    d
01DB E5                 push    h
01DC EB                 xchg
01DD 215C03             lxi     h,hexstr        ;where to build string
01E0 7A                 mov     a,d
01E1 CDF301             call    byte    ;get A as 2 ASCII chars at (HL)
01E4 7B                 mov     a,e
01E5 CDF301             call    byte    ;again, low byte
01E8 115C03             lxi     d,hexstr
01EB 0E09               mvi     c,printf
01ED CD0500             call    bdos    ;print it
01F0 E1                 pop     h
01F1 D1                 pop     d
01F2 C9                 ret
               ;
               ; Convert byte to hex ASCII chars, put at (HL)
01F3 F5        byte:    push    psw
01F4 1F                 rar             ;get high nybble
01F5 1F                 rar
01F6 1F                 rar
01F7 1F                 rar
01F8 CDFC01             call    nybble
01FB F1                 pop     psw     ;fall through for low nybble
               ; nybble makes 1 char, advances output pointer
01FC E60F      nybble: ani      0fh
01FE C630               adi     '0'
0200 FE3A               cpi     3ah
0202 DA0702             jc      nput
0205 C607               adi     7
0207 77        nput:    mov     m,a
0208 23                 inx     h
0209 C9                 ret
               ;
               ; Acceptable command inputs go in this table
               alphtbl:
020A 00                 db      0       ;dummy for no match
020B 41                 db      'A'
020C 47                 db      'G'
020D 4C                 db      'L'
020E 50                 db      'P'
020F 51                 db      'Q'
               ;
                        if      alone
                        db      'R'
                        endif   ;alone
               ;
0210 58                 db      'X'
0007 =         altblen:         equ     $-alphtbl
               ; addresses of action routines, same order as alphtabl
               addrtbl:
0211 6701               dw      na      ;not available
0213 1F01               dw      adset   ;address set
0215 6E00               dw      getbank
0217 4701               dw      lnset   ;length set
0219 B100               dw      putbank
021B 1201               dw      quit
               ;
                        if      alone
                        dw      remove
                        endif
               ;
```

(Continued on page 50)

Listing One

```
 021D 5F01              dw      xpert   ;expert mode, no menu
                ;
 021F 42616E6B73menu:   db      'Bankswap 1.0 by A. S. Woodhull
10/20/83',0dh,0ah
 0248 66756E6374        db      'functions available:',0dh,0ah
 025E 09412E2E2E        db      '       A...set move Addresses',0dh,0ah
 0277 09472E2E2E        db      '       G...Get alternate bank',0dh,0ah
 0290 094C2E2E2E        db      '       L...set move Length',0dh,0ah
 02A6 09502E2E2E        db      '       P...Put alternate bank',0dh,0ah
 02BF 09512E2E2E        db      '       Q...Quit to SID or DDT',0dh,0ah
                ;
                        if      alone
                        db      '       R...Remove BANKSWAP',0dh,0ah
                        endif
                ;
 02D8 09582E2E2E        db      '       X...eXpert mode (no menu)',0dh,0ah
 02F4 0D0A3F3F20query:  db      0dh,0ah,'?? $'
 02FA 2E2E2E6675namsg:  db      '...function not available.'
                crlfstring:
 0314 0D0A24            db      0dh,0ah,'$'
 0317 42616E6B20b0id:   db      'Bank 0 addr: $'
 0325 42616E6B20blid:   db      'Bank 1 addr: $'
 0333 4C656E6774lnmsg:  db      'Length is now $'
                chquery:
 0342 4368616E67        db      'Change to? (CR to keep): $'
 035C           hexstr  ds      4
 0360 480D0A24          db      'H',0dh,0ah,'$'
 0364 52652D656Eqmsg:   db      'Re-enter BANKSWAP from DDT or SID by "G28"'
 038E 0D0A24            db      0dh,0ah,'$'
                ;
                quietflag:
 0391 00                db      0       ;initialized off

 0392 08        inbuf:  db      8       ;max length of buffer
 0393                   ds      9
                ;
                ; default parameters: alter by Set and Length commands
 039C 0001      b0start:        dw      b0dft
 039E 0001      blstart:        dw      bldft
 03A0 0010      length:         dw      movcnt
                ;
                ; One-time routine, on 1st BDOS call intercepted
                install:
 03A2 C5                push    b       ;keep everything as it was
 03A3 D5                push    d       ;...so BDOS function can be
 03A4 E5                push    h       ;...completed
 03A5 F5                push    psw
                ; set up restart vector for re-entry
 03A6 3EC3              mvi     a,0c3h  ;a JMP instruction
 03A8 322800            sta     rstv
 03AB 211B00            lxi     h,bankexam
 03AE 222900            shld    rstv+1
                ; set up address of BIOS routine accessed directly
 03B1 2A0100            lhld    biosv           ;find where BIOS is
 03B4 114E00            lxi     d,selmem        ;...and add offset
 03B7 19                dad     d
 03B8 220804            shld    selmv
                ; then patch the RSX prefix to prevent reinstallation
 03BB 2A0A00            lhld    next+1
 03BE 220700            shld    start+1
                ; tell 'em we're ready
 03C1 0E09              mvi     c,printf
 03C3 11D003            lxi     d,imsg
 03C6 CD0900            call    next
 03C9 F1                pop     psw     ;continue with the task that
 03CA E1                pop     h       ;...was so rudely interrupted
 03CB D1                pop     d
 03CC C1                pop     b
 03CD C30900            jmp     next
```

(Continued on page 52)

Listing One

```
               ;
03D0 42414E4B53imsg:    db      'BANKSWAP loaded. To access from DDT or '
03F7 5349442074         db      'SID type "G28"'
0405 0D0A24             db      0dh,0ah,'$'
               ;
               ; reuse installation code area for stack space
               locstk:
               ;
               ; uninitialized storage
0408           selmv:  ds      2       ;used for BIOS call to selmem
040A           source: ds      2       ;for moves to buffer
040C           dest:   ds      2       ;for moves from buffer
040E           count:  ds      1       ;blocks to move
040F           holdsp: ds      2       ;stack pointer from DDT or SID
               ;
               buffer:
0411                   ds      buflen
0511                   end
```

End Listing One

Listing Two

```
; RSXMAKER.SUB assemble RSX and attach to existing COM file
; ASW 21 Oct 83
; Usage: A>SUBMIT RSXMAKER <comfile> <asmfile>
;
rmac $2
link $2 [op]
ren $2.rsx=$2.prl
gencom $1 $2
```

End Listing Two

Listing Three

```
Listing Three. An example of the use of BANKSWAP to allow examination of a
        portion of a custom disk drive routine installed in bank 0.
        (Comments added in parentheses).

D>sid
BANKSWAP loaded. To access from DDT or SID type "G28"
CP/M 3 SID - Version 3.0
#g28                                    (go to BANKSWAP)
Bankswap 1.0 by A. S. Woodhull 10/20/83
functions available:
        A...set move Addresses
        G...Get alternate bank
        L...set move Length
        P...Put alternate bank
        Q...Quit to SID or DDT
        X...eXpert mode (no menu)
?? x                                    (remove the menu)
?? a                                    (set up range)
Bank 0 addr: 0100H
Change to? (CR to keep): 7000           (don't accept default)
Bank 1 addr: 0100H
Change to? (CR to keep): 7000
Length is now 1000H
Change to? (CR to keep): <CR>           (length is OK)
?? g                                    (get the data from bank 0)
?? q                                    (back to SID)
Re-enter BANKSWAP from DDT or SID by "G28"
*D61E                                   (we're back)
#d7c00 7c3f                             (examine the data from bank 0)
7C00: C3 09 7C C3 0F 7C C3 15 7C 21 94 0C C3 00 73 21 ..|..|..|!....s!
7C10: 74 0C C3 00 73 21 1B 0C C3 00 73 AE F1 0C A9 04 t...s!....s.....
7C20: D0 2D AE F1 0C AC DF 0C B9 BA 0C 9D 8E C0 AD DE .-..............
7C30: 0C 4A 48 A9 00 2A 0D EF 0C 0D F1 0C A8 B9 80 C0 .JH..*..........
#^C                                     (return to CP/M)
D>
```

End Listings

Adding a COPY Command to ProDOS

by Shawn Day

If you use ProDOS on an Apple or Apple-compatible system, you may have noticed that you cannot copy files from one volume to another without running FILER and losing any BASIC program you have in memory. I ran across this problem when I tried to copy files to the ProDOS RAM disk from within a BASIC program: I wanted my startup program to load all of the ProDOS editor and assembler files into the RAM disk so that I could have the maximum amount of storage space on my single disk drive and speed up the process of assembling long source files. To solve this problem, I wrote a program that adds a COPY command to ProDOS.

Additional banks of memory can buy you both speed and more usable memory even though your program must reside entirely in one bank.

Using the Program

To add the COPY command to ProDOS, simply type BRUN COPY or use the general-purpose "run" command, -COPY. The code will be installed in memory above the ProDOS file buffers and protected in the system bit map. To execute the command, use the following syntax:

```
COPY pathname1,pathname2
```

A new file (pathname2) will be created, and the contents of the file indicated by pathname1 will be copied into it. You can use this command to make backup copies of any file on the same volume or to copy files from one volume to another. If you are in the process of copying files from one volume to another and no ProDOS prefix has been set, you must specify the complete path name for both files, e.g.,

```
COPY /MYDISK/HELLO,
   /RAM/STARTUP
```

If a ProDOS prefix has been set, it will automatically be prefixed to either path name if required. For example, if the prefix is /RAM/, the command

```
COPY /MYDISK/HELLO,
   STARTUP
```

will copy the file /MYDISK/HELLO to the file /RAM/STARTUP. If the destination file name already exists on the destination volume, the command exits with the DUPLICATE FILENAME error. This prevents accidental erasure of files. If you want to copy a file over an already existing file, use the DELETE command to destroy the existing file and then use the COPY command.

COPY cannot be used to copy files directly from one disk to another if you have only one disk drive. However, if you have a ProDOS RAM disk available to you (that is, you are using an Apple IIc or a IIe with an extended 80-column card) you have the capability to copy the file into the RAM disk and then copy the file again from the RAM disk over to the destination disk.

Shawn Day, 724 Glenburn St., Kelowna, British Columbia V1Y 4G6 Canada

Entering the Program

The source code for COPY is shown in Listing One (page 58). To type in the program, you can either use an assembler or type the hexadecimal numbers in from the monitor. When you are finished, save the program with

```
BSAVE COPY,A$6000,L$2B6
```

How it Works

The COPY code is divided logically into three parts: installation, first-stage processing, and second-stage processing. I will describe the operation of each in turn.

Installation

When COPY is invoked, it relocates itself to reside between the ProDOS BASIC.SYSTEM and its file buffers. A call to GETBUFR with the accumulator containing the value $02 (Listing One, lines 69 and 70) reserves the required two pages of memory. The high byte of the starting address of the reserved space is returned in the accumulator. Line 71 checks to see if the call to GETBUFR was successful. In ProDOS Version 1.0.1 there is a bug in GETBUFR that causes it to return a successful error code whether or not an error occurs. Due to this bug, COPY will only work with ProDOS Version 1.1.1 and later. If you have an earlier version of ProDOS, see your dealer to get an update. If GETBUFR is unsuccessful, the program exits with an appropriate error message (line 72).

Assuming the GETBUFR call was successful, program control is transferred to GOTSPACE (lines 73–83), which protects the acquired memory space via the system bit map and updates RSHIMEM. The routine BITMAPS (lines 135–157) is called twice—once to protect each page. Then the value of RSHIMEM in the BASIC.SYSTEM's global page is decreased by 2 (lines 80–83). This protects the acquired memory from a call to FREBUFR. FREBUFR resets HIMEM to the value contained in RSHIMEM. Because COPY dynamically allocates buffer space during execution via GETBUFR and FREBUFR, it is necessary to perform this

step. If you wish to have any other added ProDOS commands resident in memory during the execution of COPY, they must each be protected from FREBUFR by this method.

The next program segment (lines 87–95) hooks the COPY command into BASIC.SYSTEM's external command link. Any previously installed external commands are daisy chained by the method described in the *ProDOS Technical Reference Manual.* Lines (99–115) update all internal, nonrelocatable references so that the program will execute properly in its final location. They make use of the table of nonrelocatable references in lines 161–188.

After all the references are adjusted, the code is moved to its final location by the monitor MOVE routine (lines 119–131). This finishes the installation of COPY, and control is passed back to the system.

First-Stage Processing

When ProDOS runs across a command that it does not recognize, it passes control to EXTRNCMD in the BASIC.SYSTEM global page. With COPY installed, this location contains a jump to the relocated copy code (starting at line 203). Here the path-name buffer is checked to see if it contains the COPY command in upper or lower case. If not, the carry flag is set to indicate no match, and control passes to the next user-installed external command (lines 214-215 and 193). If there are no more user-installed commands, line 193 will contain a jump to an RTS instruction. Assuming the COPY command was found, execution continues at line 218, where the command-string length is decremented and stored in XLEN. Lines 220 and 221 store a 0 in XCNUM to indicate to BASIC.SYSTEM that the command is external, and lines 222–225 store the address where second-stage processing is to be resumed after the input string has been parsed by BASIC.SYSTEM. Next, PBITS and PBITS+1 are set to indicate that file creation is allowed, that the command requires two path names, and that the current prefix should be fetched if none is specified in the command string. Finally, the program exits with the carry flag cleared to tell BASIC.SYSTEM that the command has been identified.

Second-Stage Processing

After BASIC.SYSTEM has parsed the command string, control is passed to line 236. At this point, FBITS and FBITS+1 have been set by BASIC.SYSTEM to indicate which parameters were present in the command string. Lines 236–243 check that two path names were given and exit with a syntax error if not.

Assuming both path names were present, execution continues at PARMSOK (line 247), where a GET_FILE_INFO call is made to the MLI (machine-language interface) through the GOSYSTEM vector. This call gets the file information for the source file (the first path name specified in the command string). The file type is checked to make sure it is not a directory or bad block file. (Copying these file types would be meaningless.) If the file is either of these two types, the program exits with a FILE TYPE MISMATCH error.

Next, a new file is created with the file name specified by the second path name in the command line (lines 262–277). If an error occurs during the CREATE attempt, control passes to BADCALL (line 277), where the MLI error is translated to a BASIC.SYSTEM error, the carry flag is set,

and an RTS instruction transfers control back to BASIC.SYSTEM. If the CREATE is successful, lines 281–289 attempt to obtain a 1K file buffer from the system for use by the destination file. If the attempt is unsuccessful, the NO BUFFERS AVAILABLE error is generated (lines 284–286).

Finally, the rest of free memory (excluding any resident BASIC programs and any variables that may have values assigned to them) is reserved for the file-transfer buffer (lines 294–316). It is desirable to have as large a buffer as possible in order to speed up the copy. If a very large BASIC program is in memory at the time the COPY command is executed, the file will be copied in several passes because only a small amount of it can be read into memory at one time. If there is no BASIC program in memory, even very large files can be copied in one or two passes. If there is less than one free page of memory, all of the buffer space obtained so far is returned to the system, and the program exits with the NO BUFFERS AVAILABLE error (lines 302–305).

Now that all the buffers are set up, lines 320–331 open the destination file, and lines 335–337 open the source file. Any errors detected during this process cause the buffer space to be returned to the system and an appropriate error message to be generated. Lines 346–351 call NEWLINE to indicate that no NEWLINE character is to be recognized. The NEWLINE character is used by ProDOS when reading a file to signify the end of a line or record. For example, when reading lines from a text file, the NEWLINE character should be set to $8D (carriage return). This allows processing of the file line by line. Because it is desirable to read in as much of the file as possible during each pass, the NEWLINE character is disabled by setting the mask byte to 0 (lines 346 and 347).

Lines 361–378 perform the actual transfer. Lines 363 and 364 call the MLI READ routine to read in as much of the file as possible. An error during the read causes program control to pass to line 366, where the end-of-file condition is checked. If the end of file has been reached, lines 382–401 close both files, free the allocated buffer space, and return control to the system. Otherwise, lines 369–375 write the buffer contents out to the new file and then (assuming no errors were detected during the WRITE) go back for more (line 376). Lines 402–432 are the parameter lists used during the calls to the MLI.

Improvements and Modifications

The most obvious improvement is to allow the use of disk-to-disk copies with a single disk drive. You might also find it useful to allow COPYing a text file to the screen or a printer. Then you would have a general-purpose copy utility, such as CP/M's PIP command.

DDJ

(Listing begins on next page)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 194**.

ProDos Copy Listing (Text begins on page 54)

```
0000:                      LST   GEN,VSYM
0000:             ***************************************************
0000:             *                                                 *
0000:             *     COPY Command for ProDOS BASIC.SYSTEM        *
0000:             *                by Shawn Day                     *
0000:             *                                                 *
0000:             *       Syntax: COPY pathname1,pathname2          *
0000:             *                                                 *
0000:             * This command copies the file specified by       *
0000:             * pathname1, to the file specified by pathname2.  *
0000:             * If pathname2 does not exist, it is created.     *
0000:             * If pathname2 does exist, the command exits      *
0000:             * with a DUPLICATE FILENAME error.                *
0000:             *                                                 *
0000:             *       Apple ProDOS Assembler version 1.0        *
0000:             *                                                 *
0000:             ***************************************************
0000:        003C A1L       EQU   $3C          ;Start address for MOVE
0000:        003D A1H       EQU   $3D
0000:        003E A2L       EQU   $3E          ;End address for MOVE
0000:        003F A2H       EQU   $3F
0000:        0042 A4L       EQU   $42          ;Destination for MOVE
0000:        0043 A4H       EQU   $43
0000:        0073 HIMEM     EQU   $73          ;HIMEM pointer
0000:        00FC TEMP      EQU   $FC          ;Temporary storage
0000:        00FD OFFSET    EQU   $FD          ;Offset for relocation code
0000:        00FE PTR       EQU   $FE          ;Temporary address pointer
0000:        00FE COUNT     EQU   $FE          ;Temporary storage
0000:        0200 INBUF     EQU   $200         ;Keyboard input buffer
0000:        BE06 EXTRNCMD  EQU   $BE06        ;External cmd jmp vector
0000:        BE09 ERROUT    EQU   $BE09        ;Print error message
0000:        BE50 XTRNADDR  EQU   $BE50        ;External cmd address
0000:        BE52 XLEN      EQU   $BE52        ;Length of cmd string-1
0000:        BE53 XCNUM     EQU   $BE53        ;cmd no. (extrn. cmd = 0)
0000:        BE54 PBITS     EQU   $BE54        ;Command parameter bits
0000:        BE56 FBITS     EQU   $BE56        ;Command parameter bits found
0000:        BE6C VPATH1    EQU   $BE6C        ;Pathname 1 buffer pointer
0000:        BE6E VPATH2    EQU   $BE6E        ;Pathname 2 buffer
0000:        BE70 GOSYSTEM  EQU   $BE70        ;Perform MLI call
0000:        BE8B BADCALL   EQU   $BE8B        ;Translate MLI error to BI error
0000:        BEB4 SSGINFO   EQU   $BEB4        ;GET_FILE_INFO MLI buffer
0000:        BEB7 FIACESS   EQU   $BEB7        ;Access used by lock/unlock
0000:        BEB8 FIFILID   EQU   $BEB8        ;File id of disk file
0000:        BEB9 FIAUXID   EQU   $BEB9        ;Aux id
0000:        BECE OSYSBUF   EQU   $BECE        ;Buffer for MLI OPEN
0000:        BED0 OREFNUM   EQU   $BED0        ;Reference number of opened file
0000:        BED2 NEWLREF   EQU   $BED2        ;NEWLINE file reference number
0000:        BED3 NLINENBL  EQU   $BED3        ;NEWLINE enable mask and char
0000:        BED6 RWREFNUM  EQU   $BED6        ;Read file reference number
0000:        BED7 RWDATA    EQU   $BED7        ;Read buffer
0000:        BED9 RWCOUNT   EQU   $BED9        ;Read (Number of char requested)
0000:        BEDB RWTRANS   EQU   $BEDB        ;Read (Number of char transferred)
0000:        BEDE CFREFNUM  EQU   $BEDE        ;Close file reference number
0000:        BEF5 GETBUFR   EQU   $BEF5        ;Get buffer from system
0000:        BEF8 FREBUFR   EQU   $BEF8        ;Give buffer back to system
0000:        BEFB RSHIMEM   EQU   $BEFB        ;HIMEM reset value for FREEBUFR
0000:        BF00 MLI       EQU   $BF00        ;MLI entry point
0000:        BF58 BITMAP    EQU   $BF58        ;System bit map table
0000:        FE2C MOVE      EQU   $FE2C        ;Monitor MOVE routine
----- NEXT OBJECT FILE NAME IS COPY
6000:        6000           ORG   $6000
6000:             *---------------------------------------------
6000:             * Installation
6000:             *---------------------------------------------
6000:             *
6000:             *         Get some space for the COPY code
6000:             *         and protect it by updating the system bit map
6000:             *
6000:AD 03 61                LDA   PAGES        ;Number of pages needed
6003:20 F5 BE                JSR   GETBUFR      ;Ask the BI for some space
6006:90 03   600B            BCC   GOTSPACE     ;We got the space
6008:4C 09 BE                JMP   ERROUT       ;Couldn't find space
600B:48           GOTSPACE   PHA                 ;Save start address
600C:AE 03 61                LDX   PAGES        ;Number of pages to protect
600F:20 73 60     PROTECT    JSR   BITMAPS      ;Protect a page
6012:18                      CLC
6013:69 01                   ADC   #$01         ;Point to next page
6015:CA                      DEX                 ;Are we done?
6016:D0 F7   600F            BNE   PROTECT      ;If not
6018:AD FB BE                LDA   RSHIMEM      ;Protect code from FREEBUFR
601B:38                      SEC
601C:ED 03 61                SBC   PAGES
601F:8D FB BE                STA   RSHIMEM
6022:             *
6022:             *         Hook routine into the ProDOS external command vector
6022:             *
6022:AE 07 BE                LDX   EXTRNCMD+1   ;Set link to next external command
6025:AC 08 BE                LDY   EXTRNCMD+2   ;or RTS (whichever was already there)
6028:8E 01 61                STX   LINK+1
602B:8C 02 61                STY   LINK+2
602E:A9 0E                   LDA   #>START      ;Install the address of the
6030:8D 07 BE                STA   EXTRNCMD+1   ;COPY command handler in the
6033:68                      PLA                 ;external command jump vector
6034:48                      PHA                 ;Save start address for MOVE
6035:8D 08 BE                STA   EXTRNCMD+2
6038:             *
6038:             *         Update all non-relocatable references in the COPY code
6038:             *
6038:38                      SEC                 ;Find offset to be added to all
6039:E9 61                   SBC   #<LINK       ;non-relocatable references
603B:85 FD                   STA   OFFSET
603D:A2 00                   LDX   #$00         ;Index start of table
603F:A0 00                   LDY   #$00
6041:18                      CLC
```

(Continued on page 60)

ProDos Copy Listing

(Listing continued, text begins on page 54)

```
6042:BD 91 60        RELOCATE  LDA   RELTABLE,X     ;Point to non-relocatable reference
6045:85 FE                     STA   PTR
6047:BD 92 60                  LDA   RELTABLE+1,X
604A:85 FF                     STA   PTR+1
604C:B1 FE                     LDA   (PTR),Y        ;Add offset to the reference
604E:65 FD                     ADC   OFFSET
6050:91 FE                     STA   (PTR),Y
6052:E8                        INX                   ;Index next reference
6053:E8                        INX
6054:EC C7 60                  CPX   TABLESIZE      ;Number of references times 2
6057:90 E9   6042              BCC   RELOCATE
6059:                *
6059:                *         Move the code to its final home
6059:                *
6059:A2 00                     LDX   #>LINK         ;Start
605B:A0 61                     LDY   #<LINK
605D:86 3C                     STX   A1L
605F:84 3D                     STY   A1H
6061:A2 AB                     LDX   #>END          ;End
6063:A0 62                     LDY   #<END
6065:86 3E                     STX   A2L
6067:84 3F                     STY   A2H
6069:A0 00                     LDY   #$00           ;Destination (Always on a page boundary)
606B:84 42                     STY   A4L
606D:68                        PLA                   ;We saved the high byte on the stack
606E:85 43                     STA   A4H
6070:4C 2C FE                  JMP   MOVE           ;Y-reg=$00
6073:                *
6073:                *         Subroutine to update system bit map
6073:                *
6073:A8              BITMAPS   TAY                   ;Save page address
6074:48                        PHA
6075:8A                        TXA                   ;Save X-reg
6076:48                        PHA
6077:98                        TYA                   ;Restore page address
6078:4A                        LSR   A              ;Divide page address by 8
6079:4A                        LSR   A
607A:4A                        LSR   A
607B:AA                        TAX                   ;Load X-reg with page address/8
607C:98                        TYA                   ;Restore page address
607D:29 07                     AND   #$07           ;Kill off the high order bits
607F:A8                        TAY                   ;and put it in the Y-reg
6080:A9 00                     LDA   #$00           ;Zero the mask
6082:38                        SEC                   ;Prepare to put a 1 into mask
6083:6A              BITMAPS1  ROR   A              ;Rotate carry into mask
6084:88                        DEY
6085:10 FC   6083              BPL   BITMAPS1       ;Rotate mask to proper position
6087:1D 58 BF                  ORA   BITMAP,X       ;Update the actual bitmap
608A:9D 58 BF                  STA   BITMAP,X
608D:68                        PLA                   ;Restore X-reg and A-reg
608E:AA                        TAX
608F:68                        PLA
6090:60                        RTS
6091:                *
6091:                *         Table of non-relocatable references
6091:                *
6091:1F 61           RELTABLE  DW    RELOC1+2
6093:24 61                     DW    RELOC2+2
6095:2C 61                     DW    NXTCHR2+2
6097:3E 61                     DW    RELOC3+1
6099:78 61                     DW    FILEOK+2
609B:7E 61                     DW    RELOC4+2
609D:84 61                     DW    RELOC5+2
609F:89 61                     DW    RELOC6+2
60A1:8F 61                     DW    RELOC7+2
60A3:95 61                     DW    RELOC8+2
60A5:9B 61                     DW    RELOC9+1
60A7:AE 61                     DW    RELOC10+2
60A9:B3 61                     DW    RELOC11+2
60AB:D6 61                     DW    RELOC12+2
60AD:DF 61                     DW    RELOC13+2
60AF:E7 61                     DW    RELOC14+2
60B1:ED 61                     DW    RELOC15+2
60B3:F3 61                     DW    RELOC16+2
60B5:F9 61                     DW    RELOC17+1
60B7:3B 62                     DW    TRANSFER+2
60B9:3E 62                     DW    RELOC18+2
60BB:51 62                     DW    RELOC19+2
60BD:57 62                     DW    RELOC20+2
60BF:5D 62                     DW    RELOC21+1
60C1:74 62                     DW    RELOC22+2
60C3:77 62                     DW    RELOC23+2
60C5:7D 62                     DW    RELOC24+1
60C7:36              TABLESIZE DFB   TABLESIZE-RELTABLE
60C8:        0038    FILLPAGE  DS    >0-FILLPAGE    ;Fill to next page boundary
6100:                *-----------------------------------------
6100:                * The following is the COPY command code.
6100:                *-----------------------------------------
6100:4C 00 00        LINK      JMP   $0000          ;Address planted to next command
6103:02              PAGES     DFB   <END-LINK+256  ;Number of pages occupied
6104:04 43 4F 50     CMDSTRU   STR   "COPY"         ;This is the command string
6108:59
6109:04 63 6F 70     CMDSTRL   STR   "copy"         ;Accept upper or lower case
610D:79
610E:                *-----------------------------------------
610E:                * Check to see if the command is COPY. If not,
610E:                * then check for next user command, or tell ProDOS
610E:                * it is not a user command. If it is the COPY
610E:                * command, then set up for first stage processing.
610E:                *-----------------------------------------
610E:AD 6C BE        START     LDA   VPATH1         ;Get command line pointer
6111:85 FE                     STA   PTR
6113:AD 6D BE                  LDA   VPATH1+1
6116:85 FF                     STA   PTR+1
```

(Continued on page 62)

Prospero's FIPS-validated Pascal language compiler system

PRO PASCAL *works*

Small or Large

Choice of small or large model 8086 code

Used at 3000 sites around the world

WE SPECIALISE IN WRITING COMPILERS

Lazy I/O
All files closed on exit from procedure

8087

Optional 8087 support included

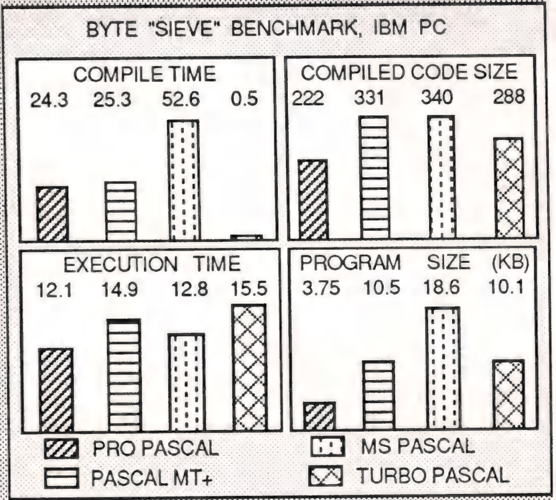
"Superb Quality"

Validated to ANSI/IEEE 770X3.97 Class A on the IBM Personal Computer - the only validated Pascal compiler available for the PC and other 8086 family machines!

NO BUGS YOU DON'T HAVE TO FILL IN ALL THE HOLES

7/16 digit Single and double precision IEEE format arithmetic

Pro Pascal allows chaining and execution of other programs.



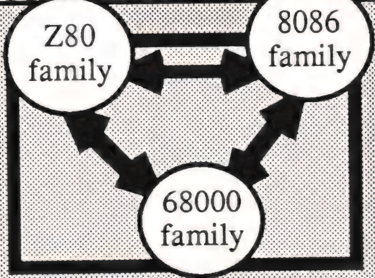
✓ Produces compact efficient code

U.S.A. Distributed by Lifeboat Ass NY and Programmer's Shop MA

Lazy Overlaying

Unique fluid overlaying technique loads non-resident routines automatically

Compiles BIG programs . . . .
>10000 lines... >1000 identifiers....
Separate compilation to build libraries and massive program suites



Portable source code

$390.00

To buy call London 01-741 8531. Visa, Amex, Mastercard & Diners accepted.

Powerful symbolic debugger included

Display and alter variables. Show source as program runs, or on request. Break on variables or procedure calls. Show calling nest.

Prospero Software

Languages For Microcomputer Professionals

190 Castelnau, London SW13 9DH, England telephone 01-741 8531 telex 8814396

Circle **no. 62** on reader service card.

ProDos Copy Listing

(Listing continued, text begins on page 54)

```
6118:A0 00              LDY  #$00        ;Index command string
611A:C8         NXTCHR  INY              ;Increment index
611B:B1 FE              LDA  (PTR),Y     ;Get character from input string
611D:D9 04 61   RELOC1  CMP  CMDSTRU,Y   ;Does it match upper case?
6120:F0 08 612A         BEQ  NXTCHR2     ;If yes
6122:D9 09 61   RELOC2  CMP  CMDSTRL,Y   ;Does it match lower case?
6125:F0 03 612A         BEQ  NXTCHR2     ;If yes
6127:38                 SEC              ;Flag match failure
6128:B0 D6 6100         BCS  LINK        ;Go to next command in the chain
612A:CC 04 61   NXTCHR2 CPY  CMDSTRU     ;Checked all characters?
612D:D0 EB 611A         BNE  NXTCHR      ;No, so check the next one
612F:88                 DEY              ;Put command length-1 in XLEN
6130:8C 52 BE           STY  XLEN
6133:A9 00              LDA  #$00        ;Mark command as external
6135:8D 53 BE           STA  XCNUM
6138:A9 4E              LDA  #>SECOND    ;Point to command handler
613A:8D 50 BE           STA  XTRNADDR
613D:A9 61      RELOC3  LDA  #<SECOND
613F:8D 51 BE           STA  XTRNADDR+1
6142:A9 0B              LDA  #$0B        ;Allow CREATE, require two pathnames
6144:8D 54 BE           STA  PBITS
6147:A9 04              LDA  #$04        ;Fetch prefix, if required
6149:8D 55 BE           STA  PBITS+1
614C:18                 CLC              ;No errors so far
614D:60                 RTS              ;Return to BASIC.SYSTEM
614E:           *------------------------------------------------
614E:           * COPY second stage processing. First check FBITS
614E:           * to see if the proper parameters were present.
614E:           *------------------------------------------------
614E:AD 56 BE   SECOND  LDA  FBITS       ;Parameters found
6151:4A                 LSR  A           ;Filename found?
6152:90 03 6157         BCC  SYNERROR    ;No, so give error message
6154:4A                 LSR  A           ;Second filename found?
6155:B0 04 615B         BCS  PARMSOK     ;Yes, so go do command
6157:A9 10      SYNERROR LDA #$10        ;Syntax error
6159:38         MLIERR  SEC              ;Set carry to show error
615A:60         RETURN  RTS              ;Return to system
615B:           *
615B:           *       Both filenames were given, so execute COPY
615B:           *
615B:A9 0A      PARMSOK LDA  #$0A        ;Set up GET_FILE_INFO call
615D:8D B4 BE           STA  SSGINFO
6160:A9 C4              LDA  #$C4        ;MLI GET_FILE_INFO code
6162:20 70 BE           JSR  GOSYSTEM    ;Perform the MLI call
6165:B0 F3 615A         BCS  RETURN      ;If error, return carry set
6167:AD B8 BE           LDA  FIFILID     ;Get file type
616A:C9 0F              CMP  #$0F        ;Can't copy a directory file
616C:F0 04 6172         BEQ  MISMATCH
616E:C9 01              CMP  #$01        ;Can't copy a bad block file
6170:D0 04 6176         BNE  FILEOK
6172:A9 0D      MISMATCH LDA #13         ;FILE TYPE MISMATCH error
6174:D0 E3 6159         BNE  MLIERR      ;Always
6176:           *
6176:           *       File type is OK
6176:           *
6176:8D 93 62   FILEOK  STA  CRFILID     ;File type for CREATE
6179:AD B9 BE           LDA  FIAUXID     ;File aux type for CREATE
617C:8D 94 62   RELOC4  STA  CRAUXID
617F:AD BA BE           LDA  FIAUXID+1
6182:8D 95 62   RELOC5  STA  CRAUXID+1
6185:A9 C3              LDA  #$C3        ;File access (UNLOCKED)
6187:8D 92 62   RELOC6  STA  CRACCESS    ;We have to be able to write!
618A:AD 6E BE           LDA  VPATH2      ;Pathname for CREATE
618D:8D 90 62   RELOC7  STA  CRPATH
6190:AD 6F BE           LDA  VPATH2+1
6193:8D 91 62   RELOC8  STA  CRPATH+1
6196:20 00 BF   CREATE  JSR  MLI         ;Create the new file
6199:C0                 DFB  $C0         ;MLI CREATE command code
619A:8F 62      RELOC9  DW   CRPARMS     ;CREATE parameter list
619C:90 03 61A1         BCC  SUCCESS     ;CREATE successful
619E:4C 8B BE           JMP  BADCALL     ;Translate to BI error code and exit
61A1:           *
61A1:           *       Create a file buffer for the destination file
61A1:           *
61A1:A9 04      SUCCESS LDA  #$04        ;1K buffer required
61A3:20 F5 BE           JSR  GETBUFR     ;Get space
61A6:90 04 61AC         BCC  RELOC10
61A8:A9 0C              LDA  #$0C        ;NO BUFFERS AVAILABLE error
61AA:38                 SEC
61AB:60                 RTS
61AC:8D 9F 62   RELOC10 STA  OPEN2BUF+1
61AF:A9 00              LDA  #$00        ;Buffer always on page boundary
61B1:8D 9E 62   RELOC11 STA  OPEN2BUF
61B4:           *
61B4:           *       Create a transfer buffer for the file
61B4:           *       Use as much free memory as possible
61B4:           *
61B4:A9 FF      TRANSBUF LDA #$FF        ;Count how many pages we can get
61B6:85 FE              STA  COUNT       ;Initialize with -1
61B8:E6 FE      GETMORE INC  COUNT       ;Count how many pages
61BA:A9 01              LDA  #$01        ;Get one page at a time
61BC:20 F5 BE           JSR  GETBUFR     ;Get space
61BF:90 F7 61B8         BCC  GETMORE     ;Keep going until we can't get more
61C1:A5 FE              LDA  COUNT       ;Check how many pages we got
61C3:D0 07 61CC         BNE  GOTENOUGH   ;Any non-zero number is okay
61C5:20 F8 BE           JSR  FREBUFR     ;Give back the allocated memory
61C8:A9 0C              LDA  #$0C        ;NO BUFFERS AVAILABLE error
61CA:38                 SEC
61CB:60                 RTS
61CC:8D DA BE   GOTENOUGH STA RWCOUNT+1  ;Save buffer size
61CF:A9 00              LDA  #$00        ;Length is always an even number of page
61D1:8D D9 BE           STA  RWCOUNT
61D4:AD 9F 62   RELOC12 LDA  OPEN2BUF+1  ;Get start address of buffer
61D7:38                 SEC
```

(Continued on page 66)

NEW !!

TURBO

NEW !!

INTEGRATED DEVELOPMENT SYSTEM

These systems were developed from the ground up to work in an integrated fashion ... This saves you time and money from trying to assemble different packages and to code in our Products Unique Features.

TURBO ISAM MASTER

GENERATES 'Ready to Run' Turbo Programs Using BORLAND'S Turbo Pascals Database TOOLBOX.

With a few key strokes you can generate the following Pascal programs/includes files. Automatically interfaces to Turbo Screen Master.

A. MASTER DATABASE PROGRAM - Generates Pascal Program Code for the following functions.

1. Add a Record • Allows both duplicate & unique only keys. • Change the up to 6 fields comprising the primary key and the up to 3 secondary keys at any time during input, and the validity of the keys are checked and the ISAM key files are automatically adjusted.

2. Delete a Record - Shows the record to be deleted on the screen and allows you to change your mind about deleting, and then adjusts the keys files automatically for the up to 4 keys. The disk space is reused automatically by the programs.

3. Edit a Record - with a key change allowed.

4. Search Database by Key - Up to 4 keys-and has a 1 key lookup to find next key in alphabetic order.

5. For Each Key - The ability to display the keys on the screen or printer in sorted order.

B. DATA BASE RECOVERY PROGRAM. This program recovers your data base if it is corrupted by a power outage or certain hardware failures.

C. MULTIPLE ISAM FILES - ·Can be used at the same time in a single program. The generator also produces context sensitive instructions on how to integrate the generated program into your main program.

D. DOCUMENTATION - Print screens and ISAM specifications. Also inline program documentation is generated.

TURBO MENU MASTER

GENERATES 'Ready to Run' Turbo Programs with Dedicated Screens or Windows.

Features: Has Menu Data Base so its easy to modify menues.
Provides Selection by - **(1) Press of a number**-If you have more than 9 selections then you enter only a single number if the selection is 2-9. **(2) Press of a function key** - If more than 10 selec tions then can enter use shift function key to get up to 18 selections. **(3) Press the High Lighted Letter** -In the selection description, you choose a letter for each menu selection in the interactive menu generator. **(4) Use the arrow keys** - to point to a menu item & then press return.

Adaptive Screen Colors -Different screen colors are automatically used for color or monochrome display. This allows you to provide a better interface for the user.

Interactive Menu Builder - (1) Allows for the automatic entry & reorder of selections. (2) Offers easy color selection. (3) Allows for choice of procedure, chain, execute or comment code generation for each selection. (4) Has startup Menu that YOU design. (5) Provides for the integrated Display & Control of the Key Lock Status.

RISK FREE TRIAL

Try the demo package included for 30 days. If not pleased return for a full refund.

NO ROYALTIES on Generated Programs

Credit card orders call:

1-800-821-9503

In Florida (800) 342-0137

HAWAIIAN VILLAGE COMPUTER SOFTWARE
1109 Pennsylvania Ave. St. Cloud, Florida 32769

Circle no. 133 on reader service card.

TURBO SCREEN MASTER

Some of the **UNIQUE** features of the **Turbo Screen Generator** are:
Intelligent Cursor - Computer automatically determines which single/double character to use from the Extended IBM Character set to draw boxes and borders. You just move the cursor where you want the border. The other Screen Systems require that you must tediously choose the correct graphic character.

Extended Variable Names - Allows the use of Structures & Arrays in your variable names and automatically generates the correct declarations and initalization routines. Also you are allowed to declare a field larger than the screen display size.

Provides Three Color Selections Methods - Easy to develop color screen on Monochrome Monitor and to develop Monochrome Screens on the Color Monitor.

Adaptive Screen - Entry colors change automatically depending on what type of screen is in use.

COMPARISON WITH OTHER PRODUCTS	TURBO SCREEN MASTER	TURBO SCREEN VER - 1.10	SCREEN SCULPTURE Ver - 1.01
Border Color Control	YES	NO	NO
You Assign Variable Names No cyptic variable names whose names depend on where they appear on screen	YES	NO	YES
Range and Date Checks	YES	NO	YES
Data Entry Valid Character Set	YES	NO	NO
Data Entry Mask Helpful for Profession Screen input & Validation	YES	NO	YES
Initalize Variables to a starting value	YES	NO	YES
Data Entry Valid String Set	YES	NO	NO
Pascal storage for type of Boolean & Integer	YES	YES	NO
Control Capa/Num Look	YES	NO	NO
Auto-Initalization of Date/Time	YES	NO	NO
User Defined Error & Message Handler	YES	NO	NO
Generated program adapts automatically to IBM Screen-Monitor Type	YES	NO	YES
Handles Function Keys	YES	NO	NO
Help Screen Procedures	YES	YES	NO
Optional ISAM Keys Screens Code Generated automatically	YES	NO	NO
Turbo Toolkit Included	YES	NO	NO
Undo Function	YES	NO	NO
Provides Running Time Display	YES	NO	NO
Integrated Data Dictionary	YES	NO	NO
Auto-start Screen for Screen Generation	YES	NO	NO
PRICE (INCLUDES SHIPPING)	$89.95	54.95	125.00

FREE TURBO TOOL KIT $44.95 VALUE

WITH ORDER OF A MASTER PRODUCT

YES, Enclosed is ____________

INTRODUCTORY OFFER

☐ Turbo Pascal 3.0	~~$69.95~~	$59.95
☐ Turbo Database Tool Box	~~$54.95~~	$49.95
☐ Turbo Menu Master	~~$69.95~~	$54.95
☐ Turbo Screen Master	~~$124.95~~	$89.95
☐ Turbo ISAM Master	~~$149.95~~	$124.95

Systems Requires: IBM PC,XT,AT or 100% Compatiable -196K. MS DOS 2.0 or Higher - 80 column screen. Turbo Pascal 3.0 - 2 DD/DS Disk Drives. ISAM Program Generator Requires Turbo Database Toolbox.

•These prices include shipping to all U.S. Cities. All foreign orders add $10 per product ordered

For Other Inquires Call (305) 892-5686

Name: ____________ **Phone:** ____________
Shipping Address: ____________
City: ____________ **State:** ____________ **Zip:** ____________
VISA or MC #: ____________ **Exp. Date:** ____________

C.O.D.'s and Purchase orders WILL NOT be accepted. Outside USA: make payment by bank draft payable in US dollars drawn on a US bank.

SERVICES

File Transfer Service: Advent provides a service beyond the ability of any format conversion software! We can transfer files between MS-DOS/PC-DOS, CP/M and other operating systems in 300 different 3 1/2", 5 1/4" and 8" formats. Includes Apple and Mac, Apricot, Data General One, Kaypro 2000, Eagle, Epson QX-10 & PX-8 (ROM), HP-150, and North Star computers.

ENGINEERING SOFTWARE

ACNAP: A stand-alone Electronics Circuit Analysis Program for use with passive and active circuits consisting of resistors, capacitors, inductors, transistors, op-amps, FETs, etc. Features menu driven and very fast processing times with circuits saved to disk for later use or editing.
ACNAP (CP/M & MS-DOS) $69.95

DCNAP: Stand-alone DC circuit analysis program for use with passive and active circuits containing resistors, voltage sources, independent and dependent current sources. Fast, menu- driven program with circuit saved to disk for later use or editing.
DCNAP (CP/M & MS-DOS) $69.95

Plotpro: Scientific graph printing program. Prints on 80 or 132 column printer. Create linear, semi-logarithmic, and full logarithmic plots with one or two Y axes in auto or forced scale.
Plotpro (CP/M & MS-DOS) $69.95

SPP: This Signal Processing Program contains an integrated set of routines which analyze linear and non-linear systems and circuits and their effects on user specified time domain waveforms. Based on a 512 point Fast Fourier Transform and its inverse. Linear processing is in frequency domain and non-linear processing is in time domain.
SPP (CP/M & MS-DOS) $69.95

SOFTWARE UTILITIES

Autodiff: File difference detector. This program finds insertions, deletions, and changes between any two files. Autodiff can mark the file, display, or print the differences, and more!
Autodiff (CP/M) $29.95

CP/M DateStamper: Automatically stamp your files with the date it is created, last read, or modified. Works without a Real Time Clock , or with many clocks currently on the market. Utilities are included to allow copying, erasing, or renaming files based on time and date. A time logging utility is included to record computer usage for business/tax purposes.
DateStamper (CP/M) $49.95

Media Master +: Read and Write up to 75 CP/M, MS-DOS & TRSDOS disk formats on your IBM or look-alike computer. ZP/EM program is bundled with Media Master to allow CP/M programs to run directly on your MS-DOS computer. An $80.00 value.
Media Master + ... $59.95

Pack and Crypt: Two program set. Pack compresses and expands files on disk to save space. Crypt encodes files to provide security for sensitive data. Both are ideal for use with modem transfers.
Pack and Crypt (CP/M & MS-DOS) $24.95

Sidekick: One of the most popular programs ever written. Use Sidekick as a calculator, notepad, appointment calendar, auto dialer, ASCII conversion table and much more. On-line help if you forget any of Sidekick's many functions.
Sidekick (MS-DOS) $54.95

SmartKey II: New Release! Same great time saver as the original, and allows compiling of definitions you set up with your word-processor! Makes every software program you use easier. Can reduce keystrokes by more than 50% by redefining any key on your keyboard to be any combination of characters or commands that you desire.
SmartKey II $49.95

SmartPrint: A powerful add-on to SmartKey, SmartPrint is a vesatile writing tool designed to give you full access to your printer's features such as wide, bold, condensed, underlined, subscript, superscript, and more. Works great with programs like WordStar and others.
SmartPrint $29.95

Uniform: Your Computer can read and write up to 80 CP/M, MS-DOS / PC-DOS & TRSDOS disk formats. Versions available for most popular CP/M and MS-DOS computers. Specify your host computer when ordering.
Uniform (CP/M & MS-DOS) $69.95

XTREE: Directory maintenance program that graphically displays subdirectories and filename paths. Complete control of your directory including delete, rename, view, list or show. A must for your IBM or compatible.
XTREE (MS-DOS) $49.95

Super Zap: Disk patch and dump program. If you have used DU, you will love this menu driven marvel!
Super Zap (CP/M) $24.95

ZP/EM: Run almost any CP/M program on your IBM or clone. Use with Media Master or Uniform to allow programs on CP/M disk formats to run directly on your IBM or compatible computer.
ZP/EM (MS-DOS) $39.95

FX, QX-10, PX-8 - Epson Corp; CP/M - DRI; MS-DOS - MicroSoft; PC-DOS - IBM Corp.; dBASE II & dBase III - Ashton-Tate; WordStar - MicroPro; UNIX - Bell Laboratories; Apple - Apple Computer Inc.; Basic Business - Advent Products Inc.

PROGRAMMING LANGUAGES

C/80 Ver 3.1: Full featured C compiler and runtime library. One of the fastest on the market. Mathpak is included for true 32 bit floating point and signed integers.
C/80 Ver. 3.1 (CP/M) $79.90

C/NIX: Operating System Enhancement for CP/M. C/NIX gives your system many features in the UNIX OS such a hierarchical directory, I/O redirection, "pipes" & "filters" and command files. Uses only 2.3K of TPA and 42K of disk. Requires CP/M 2.x.
C/NIX $59.95

LISP/80: Experiment with the artificial intelligence language. Based on the INTERLISP dialect, LISP/80 offers over 75 built-in functions, including file I/O, and string operations. Complete with 36 page manual and demo programs.
LISP/80 .... $39.95

Toolworks C: This compiler is a complete subset of C. The two- pass compiler produces relocatable object files (.obj) which are compatible with the MS-DOS LINK program. Mathpak is included for true 32 bit floating point and signed integers.
Toolworks C Compiler (MS-DOS) $79.90

Turbo Pascal: Borland version 3.0. The best Pascal compiler on the market.
Turbo Pascal (CP/M & MS-DOS) $69.95

Turbo Toolbox: Set of 3 utilities for use with Turbo Pascal.
Turbo Toolbox (CP/M & MS-DOS) $54.95

Turbo Tutor: Teaches step-by-step how to use Turbo Pascal.
Turbo Tutor (CP/M & MS-DOS) $34.95

Turbo Graphics: Provides full graphics management for producing windows, pie and pie charts, circles and other geometric shapes with Turbo Pascal.
Turbo Graphics (MS-DOS) $54.95

TEXT EDITING

Punctuation & Style: Improves your writing by catching unbalanced quotes, parentheses and brackets, improper abbreviations, capitalization, sentence structure, much more. It's like having your own copy editor!
Punctuation & Style (CP/M & MS-DOS) $125.00

Word Finder: This powerful 90,000 word Thesaurus allows you to select the best word for the application. Works inside WordStar for greater ease of use. Instantly searches its dictionary, then displays synonyms, and automatically deletes the "wrong" word and replaces it with the "right" word. Requires 380K disk storage.
Word Finder (CP/M) $79.95

Wordpatch: Print files with tiny, compressed, wide, or wide compressed type faces, 5 sizes of italic, real superscripts and subscripts, and 6, 7, and 8 lines per inch spacing. No new print controls to learn. Supports most popular dot matrix printers. A must for WordStar users!
Wordpatch (CP/M & MS-DOS) $49.95

The Word Plus: The ultimate spelling checker. Not only finds misspelled words but shows you correct spelling options, shows the word in context, allows you to build dictionaries of special words you use, and much more.
The Word Plus (MS-DOS) $150.00

HARDWARE & SUPPLIES

Finger Print "Letter Writer": Unleash your Epson FX series printer. Add near-letter-quality print, IBM and/or Apple Graphics printer emulation, plus 16 other print functions! Three replacement chips quickly fit inside Epson FX series printers. Easy installation. Does not void printer warranty.
Finger Print "LetterWriter" $79.95

Diskettes, Double Density:

Maxell 10-pack w/ storage box:		3M box of 10:	
Single Sided	**$19.95**	**Single Sided**	**$22.95**
Double Sided	**$23.95**	**Double Sided**	**$26.95**

Economy Diskettes: package of 25 including tyvek sleeves.
Single Sided $29.50
Double Sided $31.25

Call or write for our FREE catalog

All items are warranteed for 90 days. 30 day money back guarantee if not completely satisfied. Guarantee for software applies only if diskette seal is intact. Visa and MasterCard are welcome. Please add 2.00 freight per total order and 2.00 for COD orders. California residents please add 6% sales tax. Prices, availability and specifications subject to change without notice.

CALL TODAY **National (800) 821-8778**
California (800) 521-7182

Hours: Mon - Fri 8 am - 5 pm PDT

DEALER INQUIRIES WELCOME.

advent products inc.
3154-F E. La Palma Ave.
Anaheim, CA 92806
(714) 630-0446

Circle no. 176 on reader service card.

ProDos Copy Listing

(Listing continued, text begins on page 54)

```
61D8:E5 FE                     SBC  COUNT
61DA:8D D8 BE                  STA  RWDATA+1      ;Save buffer start address
61DD:8D A4 62       RELOC13    STA  WR2BUF+1
61E0:A9 00                     LDA  #$00          ;Buffer starts on a page boundary
61E2:8D D7 BE                  STA  RWDATA
61E5:8D A3 62       RELOC14    STA  WR2BUF
61E8:               *
61E8:               *          Open the destination file
61E8:               *
61E8:AD 6E BE                  LDA  VPATH2        ;Pathname pointer
61EB:8D 9C 62       RELOC15    STA  OPEN2PATH
61EE:AD 6F BE                  LDA  VPATH2+1
61F1:8D 9D 62       RELOC16    STA  OPEN2PATH+1
61F4:20 00 BF                  JSR  MLI           ;Perform MLI call
61F7:C8                        DFB  $C8           ;MLI OPEN command code
61F8:9B 62          RELOC17    DW   OPEN2PARM
61FA:90 08     6204            BCC  OPENSRC       ;No errors, so open the source file
61FC:48                        PHA                 ;Save error code
61FD:20 F8 BE                  JSR  FREBUFR       ;Release buffer
6200:68                        PLA                 ;Restore error code
6201:4C 8B BE                  JMP  BADCALL       ;Translate error code and exit
6204:               *
6204:               *          Open source file
6204:               *
6204:A5 74          OPENSRC    LDA  HIMEM+1       ;Get buffer start
6206:8D CF BE                  STA  OSYSBUF+1     ;(use general purpose buffer)
6209:A9 00                     LDA  #$00          ;Buffer always on a page boundary
620B:8D CE BE                  STA  OSYSBUF
620E:A9 C8                     LDA  #$C8          ;MLI open code
6210:20 70 BE                  JSR  GOSYSTEM      ;Perform the MLI call
6213:B0 1D     6232            BCS  ERRORET       ;Exit if error detected
6215:AD D0 BE                  LDA  OREFNUM       ;Set up file reference number
6218:8D D2 BE                  STA  NEWLREF       ;For NEWLINE call
621B:8D D6 BE                  STA  RWREFNUM      ;For READ call
621E:8D DE BE                  STA  CFREFNUM      ;For CLOSE call
6221:A9 00                     LDA  #$00          ;Set up NEWLINE call
6223:8D D3 BE                  STA  NLINENBL      ;to disable NEWLINE char
6226:A9 0D                     LDA  #$0D          ;Non-zero
6228:8D D4 BE                  STA  NLINENBL+1
622B:A9 C9                     LDA  #$C9          ;MLI NEWLINE code
622D:20 70 BE                  JSR  GOSYSTEM      ;Perform the MLI call
6230:90 07     6239            BCC  TRANSFER      ;If no errors detected
6232:48             ERRORET    PHA                 ;Save error code
6233:20 65 62                  JSR  CLEANUP       ;Close files and free buffer
6236:68                        PLA                 ;Restore error code
6237:38                        SEC                 ;Flag error
6238:60                        RTS                 ;Return to system
6239:               *
6239:               *          Both files are now open so perform transfer
6239:               *
6239:AD A0 62       TRANSFER   LDA  OPEN2REF      ;Set up WRITE file reference number
623C:8D A2 62       RELOC18    STA  WR2REF
623F:A9 CA          XFERLOOP   LDA  #$CA          ;MLI READ code
6241:20 70 BE                  JSR  GOSYSTEM      ;Perform the MLI call
6244:90 06     624C            BCC  XFEROK        ;No errors
6246:C9 05                     CMP  #$05          ;End of file?
6248:F0 1B     6265            BEQ  CLEANUP       ;Yes, so clean up and exit
624A:D0 E6     6232            BNE  ERRORET       ;Exit with appropriate error
624C:AD DB BE       XFEROK     LDA  RWTRANS       ;Number of bytes transferred
624F:8D A5 62       RELOC19    STA  WR2REQ        ;Request to write these bytes
6252:AD DC BE                  LDA  RWTRANS+1
6255:8D A6 62       RELOC20    STA  WR2REQ+1
6258:20 00 BF                  JSR  MLI           ;Write the data to the destination file
625B:CB                        DFB  $CB           ;MLI WRITE code
625C:A1 62          RELOC21    DW   WR2PARMS
625E:90 DF     623F            BCC  XFERLOOP
6260:20 8B BE                  JSR  BADCALL       ;Translate to BI error code
6263:B0 CD     6232            BCS  ERRORET       ;Always
6265:               *------------------------------------------------
6265:               * Close open files and free the allocated buffer space
6265:               *------------------------------------------------
6265:A9 00          CLEANUP    LDA  #$00          ;Default to no errors detected
6267:85 FC                     STA  TEMP
6269:A9 CC                     LDA  #$CC          ;MLI CLOSE code
626B:20 70 BE                  JSR  GOSYSTEM      ;Close source file
626E:90 02     6272            BCC  RELOC22       ;If no error detected
6270:85 FC                     STA  TEMP          ;Update error
6272:AD A0 62       RELOC22    LDA  OPEN2REF      ;Close destination file
6275:8D AA 62       RELOC23    STA  CLOSE2REF
6278:20 00 BF                  JSR  MLI           ;Perform the MLI call
627B:CC                        DFB  $CC           ;MLI CLOSE code
627C:A9 62          RELOC24    DW   CL2PARMS      ;CLOSE parameter list
627E:90 05     6285            BCC  FREE          ;If no error detected
6280:20 8B BE                  JSR  BADCALL       ;Translate to BI error
6283:85 FC                     STA  TEMP          ;Update error
6285:20 F8 BE       FREE       JSR  FREBUFR       ;Deallocate file buffer
6288:18                        CLC                 ;Default to no errors detected
6289:A5 FC                     LDA  TEMP          ;Error code in A
628B:F0 01     628E            BEQ  RET           ;Zero if no errors
628D:38                        SEC                 ;Flag error
628E:60             RET        RTS
628F:               *------------------------------------------------
628F:               * Parameter list for CREATE
628F:               *------------------------------------------------
628F:07             CRPARMS    DFB  $07           ;Parameter count
6290:          0002 CRPATH     DS   2             ;Pointer to pathname
6292:          0001 CRACCESS   DS   1             ;Access permitted byte
6293:          0001 CRFILID    DS   1             ;File type
6294:          0002 CRAUXID    DS   2             ;File aux type
6296:01                        DFB  $01           ;Standard file (Not directory)
6297:00 00          CRDATE     DFB  $00,$00       ;Create date
6299:00 00          CRTIME     DFB  $00,$00       ;Create time
629B:               *------------------------------------------------
629B:               * Parameter list for OPEN
629B:               *------------------------------------------------
```

(Continued on page 68)

ProDos Copy Listing

(Listing continued, text begins on page 54)

```
629B:03            OPEN2PARM DFB   $03             ;Parameter count
629C:         0002 OPEN2PATH DS    2               ;Pointer to pathname
629E:         0002 OPEN2BUF  DS    2               ;Pointer to I/O buffer
62A0:         0001 OPEN2REF  DS    1               ;File reference number
62A1:              *--------------------------------------------------
62A1:              * Parameter list for WRITE
62A1:              *--------------------------------------------------
62A1:04            WR2PARMS  DFB   $04             ;Parameter count
62A2:         0001 WR2REF    DS    1               ;File reference number
62A3:         0002 WR2BUF    DS    2               ;File transfer buffer
62A5:         0002 WR2REQ    DS    2               ;Number of bytes to transfer
62A7:         0002 WR2TRANS  DS    2               ;Number of bytes transferred
62A9:              *--------------------------------------------------
62A9:              * Parameter list for CLOSE
62A9:              *--------------------------------------------------
62A9:01            CL2PARMS  DFB   $01             ;Parameter count
62AA:         0001 CLOSE2REF DS    1               ;File reference number
62AB:         62AB END       EQU   *


   3D A1H            3C A1L            3F A2H            3E A2L
   43 A4H            42 A4L          BE8B BADCALL      6083 BITMAPS1
 6073 BITMAPS      BF58 BITMAP       BEDE CFREFNUM     62A9 CL2PARMS
 6265 CLEANUP      62AA CLOSE2REF    6109 CMDSTRL      6104 CMDSTRU
   FE COUNT        6292 CRACCESS     6294 CRAUXID     ?6297 CRDATE
?6196 CREATE       6293 CRFILID      628F CRPARMS      6290 CRPATH
?6299 CRTIME       62AB END          6232 ERRORET      BE09 ERROUT
 BE06 EXTRNCMD     BE56 FBITS       ?BEB7 FIACESS      BEB9 FIAUXID
 BEB8 FIFILID      6176 FILEOK       60C8 FILLPAGE     BEF8 FREBUFR
 6285 FREE         BEF5 GETBUFR      61B8 GETMORE      BE70 GOSYSTEM
 61CC GOTENOUGH    600B GOTSPACE       73 HIMEM       ?0200 INBUF
 6100 LINK         6172 MISMATCH     BF00 MLI          6159 MLIERR
 FE2C MOVE         BED2 NEWLREF      BED3 NLINENBL     611A NXTCHR
 612A NXTCHR2        FD OFFSET       629E OPEN2BUF     629B OPEN2PARM
 629C OPEN2PATH    62A0 OPEN2REF     6204 OPENSRC      BED0 OREFNUM
 BECE OSYSBUF      6103 PAGES        615B PARMSOK      BE54 PBITS
 600F PROTECT        FE PTR          61AC RELOC10      61B1 RELOC11
 61D4 RELOC12      61E5 RELOC14      611D RELOC1       61DD RELOC13
 61EB RELOC15      61F1 RELOC16      61F8 RELOC17      623C RELOC18
 624F RELOC19      6255 RELOC20      625C RELOC21      6272 RELOC22
 6275 RELOC23      6122 RELOC2       627C RELOC24      613D RELOC3
 617C RELOC4       6182 RELOC5       6187 RELOC6       618D RELOC7
 6193 RELOC8       619A RELOC9       6042 RELOCATE     6091 RELTABLE
 628E RET          615A RETURN       BEFB RSHIMEM      BED9 RWCOUNT
 BED7 RWDATA       BED6 RWREFNUM     BEDB RWTRANS      614E SECOND
 BEB4 SSGINFO      610E START        61A1 SUCCESS      6157 SYNERROR
 60C7 TABLESIZE      FC TEMP        ?61B4 TRANSBUF     6239 TRANSFER
 BE6C VPATH1       BE6E VPATH2       62A3 WR2BUF       62A1 WR2PARMS
 62A2 WR2REF       62A5 WR2REQ      ?62A7 WR2TRANS     BE53 XCNUM
 623F XFERLOOP     624C XFEROK       BE52 XLEN         BE50 XTRNADDR

   3C A1L            3D A1H            3E A2L            3F A2H
   42 A4L            43 A4H              73 HIMEM          FC TEMP
   FD OFFSET         FE PTR            FE COUNT       ?0200 INBUF
 600B GOTSPACE     600F PROTECT      6042 RELOCATE     6073 BITMAPS
 6083 BITMAPS1     6091 RELTABLE     60C7 TABLESIZE    60C8 FILLPAGE
 6100 LINK         6103 PAGES        6104 CMDSTRU      6109 CMDSTRL
 610E START        611A NXTCHR       611D RELOC1       6122 RELOC2
 612A NXTCHR2      613D RELOC3       614E SECOND       6157 SYNERROR
 6159 MLIERR       615A RETURN       615B PARMSOK      6172 MISMATCH
 6176 FILEOK       617C RELOC4       6182 RELOC5       6187 RELOC6
 618D RELOC7       6193 RELOC8      ?6196 CREATE       619A RELOC9
 61A1 SUCCESS      61AC RELOC10      61B1 RELOC11     ?61B4 TRANSBUF
 61B8 GETMORE      61CC GOTENOUGH    61D4 RELOC12      61DD RELOC13
 61E5 RELOC14      61EB RELOC15      61F1 RELOC16      61F8 RELOC17
 6204 OPENSRC      6232 ERRORET      6239 TRANSFER     623C RELOC18
 623F XFERLOOP     624C XFEROK       624F RELOC19      6255 RELOC20
 625C RELOC21      6265 CLEANUP      6272 RELOC22      6275 RELOC23
 627C RELOC24      6285 FREE         628E RET          628F CRPARMS
 6290 CRPATH       6292 CRACCESS     6293 CRFILID      6294 CRAUXID
?6297 CRDATE      ?6299 CRTIME       629B OPEN2PARM    629C OPEN2PATH
 629E OPEN2BUF     62A0 OPEN2REF     62A1 WR2PARMS     62A2 WR2REF
 62A3 WR2BUF       62A5 WR2REQ      ?62A7 WR2TRANS     62A9 CL2PARMS
 62AA CLOSE2REF    62AB END          BE06 EXTRNCMD     BE09 ERROUT
 BE50 XTRNADDR     BE52 XLEN         BE53 XCNUM        BE54 PBITS
 BE56 FBITS        BE6C VPATH1       BE6E VPATH2       BE70 GOSYSTEM
 BE8B BADCALL      BEB4 SSGINFO     ?BEB7 FIACESS      BEB8 FIFILID
 BEB9 FIAUXID      BECE OSYSBUF      BED0 OREFNUM      BED2 NEWLREF
 BED3 NLINENBL     BED6 RWREFNUM     BED7 RWDATA       BED9 RWCOUNT
 BEDB RWTRANS      BEDE CFREFNUM     BEF5 GETBUFR      BEF8 FREBUFR
 BEFB RSHIMEM      BF00 MLI          BF58 BITMAP       FE2C MOVE
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 15-JAN-84 21:28
** TOTAL LINES ASSEMBLED   433
** FREE SPACE PAGE COUNT   79
```

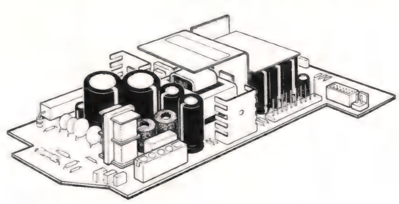
End Listing

ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A COPY OF THIS WARRAN...

DIGITAL RESEARCH COMPUTERS

(214) 225-2309

Switching Power Supply!



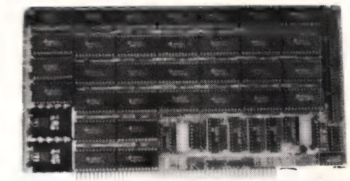
+ 5VDC - 8 Amps
+12VDC - 5 Amps
-12VDC - 1 Amp
(81 WATTS MAX)

$29.95 ea. | 4 FOR $99 | ADD $2 EA. UPS

BRAND NEW UNITS, MFG. BY BOSCHERT FOR HEWLETT PACKARD! PERFECT FOR SMALL COMPUTER AND DISK DRIVE APPLICATIONS #XL81-5630. INPUT 110V/220V, 50/60 HZ. NOMINAL OUTPUTS: +5VDC @ 8A, +12VDC @ 5A, -12VDC @ 1A. TOTAL MAX OUTPUT. MUST BE LIMITED TO 81 WATTS TOTAL! (WITH PIN OUT SHEET.) ORIGINAL FACTORY BOXED.

64K S100 STATIC RAM

$119.00 KIT



LOW POWER!
150 NS ADD $10

BLANK PC BOARD WITH DOCUMENTATION $49.95

SUPPORT ICs + CAPS $17.50

FULL SOCKET SET $14.50

FULLY SUPPORTS THE NEW IEEE 696 S100 STANDARD (AS PROPOSED)

FOR 56K KIT $105

ASSEMBLED AND TESTED ADD $50

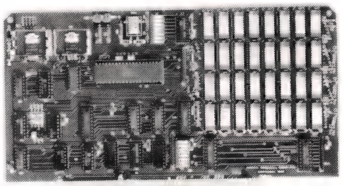
FEATURES: *PRICE CUT!*

* Uses new 2K x 8 (TMM 2016 or HM 6116) RAMs.
* Fully supports IEEE 696 24 BIT Extended Addressing.
* 64K draws only approximately 500 MA.
* 200 NS RAMs are standard. (TOSHIBA makes TMM 2016s as fast as 100 NS. FOR YOUR HIGH SPEED APPLICATIONS.)
* SUPPORTS PHANTOM (BOTH LOWER 32K AND ENTIRE BOARD).
* 2716 EPROMs may be installed in any of top 48K.
* Any of the top 8K (E000 H AND ABOVE) may be disabled to provide windows to eliminate any possible conflicts with your system monitor, disk controller, etc.
* Perfect for small systems since BOTH RAM and EPROM may co-exist on the same board.
* BOARD may be partially populated as 56K.

256K S-100 SOLID STATE DISK SIMULATOR!

WE CALL THIS BOARD THE "LIGHT-SPEED-100" BECAUSE IT OFFERS AN ASTOUNDING INCREASE IN YOUR COMPUTER'S PERFORMANCE WHEN COMPARED TO A MECHANICAL FLOPPY DISK DRIVE.

PRICE CUT!



FEATURES:
* 256K on board, using + 5V 64K DRAMS.
* Uses new Intel 8203-1 LSI Memory Controller.
* Requires only 4 Dip Switch Selectable I/O Ports.
* Runs on 8080 or Z80 S100 machines.
* Up to 8 LS-100 boards can be run together for 2 Meg. of On Line Solid State Disk Storage.
* Provisions for Battery back-up.
* Software to mate the LS-100 to your CP/M* 2.2 DOS is supplied.
* The LS-100 provides an increase in speed of up to 7 to 10 times on Disk Intensive Software.
* Compare our price! You could pay up to 3 times as much for similar boards.

BLANK PCB (WITH CP/M* 2.2 PATCHES AND INSTALL PROGRAM ON DISKETTE) $69.95
(8203-1 INTEL $29.95)

(ADD $50 FOR A&T) **$149.00**
#LS-100 (FULL 256K KIT)

PANASONIC

Green Screen - Video Monitors

25 MHZ. TYPICAL BANDWIDTH!!!

Brand New In The Box! 9-Inch Screen

#K-904B1 (Chassis #Y08A) Open Frame Style

$29.95 EA.

GROUP SPECIAL: **4 for $99.00** WITH DATA & SCHEMATIC

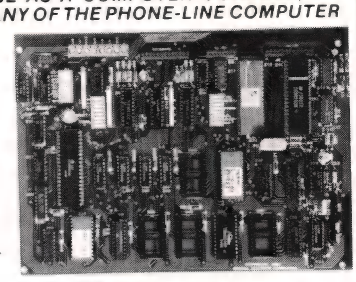
(USA SHIPPING: $3. PER UNIT. CANADA: $7. PER UNIT)

COMPUTER MANUFACTURER'S EXCESS. STILL IN ORIGINAL PANASONIC BOXES. THE CRT TUBE ALONE WOULD COST MORE THAN OUR PRICE FOR THE COMPLETE UNIT. FOR SPLIT VIDEO (TTL INPUTS) OPERATION, NOT COMPOSITE VIDEO. OPERATES FROM 12VDC AT 1 AMP. VERTICAL INPUT IS 49 TO 61 HZ. HORIZONTAL INPUT: 15,750 HZ ± 500 HZ. RESOLUTION IS 800 LINES AT CENTER 650 LINES AT CORNERS.

THE NEW ZRT-80

CRT TERMINAL BOARD!

A LOW COST Z-80 BASED SINGLE BOARD THAT ONLY NEEDS AN ASCII KEYBOARD, POWER SUPPLY, AND VIDEO MONITOR TO MAKE A COMPLETE CRT TERMINAL. USE AS A COMPUTER CONSOLE, OR WITH A MODEM FOR USE WITH ANY OF THE PHONE-LINE COMPUTER SERVICES.



FEATURES:
* Uses a Z80A and 6845 CRT Controller for powerful video capabilities.
* RS232 at 16 BAUD Rates from 75 to 19,200.
* 24 x 80 standard format (60 Hz).
* Optional formats from 24 x 80 (50 Hz) to 64 lines x 96 characters (60 Hz).
* Higher density formats require up to 3 additional 2K x 8 6116 RAMS.
* Uses N.S. INS 8250 BAUD Rate Gen. and USART combo IC.
* 3 Terminal Emulation Modes which are Dip Switch selectable. These include the LSI-ADM3A, the Heath H-19, and the Beehive.
* Composite or Split Video.
* Any polarity of video or sync.
* Inverse Video Capability.
* Small Size: 6.5 x 9 inches.
* Upper & lower case with descenders.
* 7 x 9 Character Matrix.
* Requires Par. ASCII keyboard.

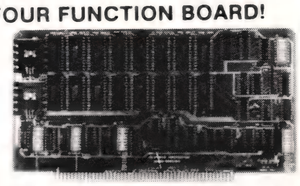
FOR 8 IN. SOURCE DISK (CP/M COMPATIBLE) ADD $10

$89.95 #ZRT-80
(COMPLETE KIT, 2K VIDEO RAM)

BLANK PCB WITH 2716 CHAR. ROM. 2732 MON. ROM $49.95
SOURCE DISKETTE - ADD $10
SET OF 2 CRYSTALS - ADD $7.50

32K S100 EPROM/STATIC RAM

NEW! FOUR FUNCTION BOARD! NEW!



EPROM II FULL EPROM KIT $69.95

A&T EPROM ADD $35.00

BLANK PC BOARD WITH DATA $39.95

SUPPORT IC'S PLUS CAPS $16

FULL SOCKET SET $15

We took our very popular 32K S100 EPROM Card and added additional logic to create a more versatile EPROM/RAM Board.

PRICES SLASHED!

FEATURES:
* This one board can be used in any one of four ways:
  A. As a 32K 2716 EPROM Board
  B. As a 32K 2732 EPROM Board (Using Every Other Socket)
  C. As a mixed 32K 2716 EPROM/2K x 8 RAM Board
  D. As a 32K Static RAM Board
* Uses New 2K x 8 (TMM2016 or HM6116) RAM's
* Fully Supports IEEE 696 Buss Standard (As Proposed)
* Supports 24 Bit Extended Adressing
* 200 NS (FAST!) RAM'S are standard on the RAM Kit
* Supports both Cromemco and North Star Bank Select
* Supports Phantom
* On Board wait State Generator
* Every 2K Block may be disabled
* Addressed as two separate 16K Blocks on any 64K Boundary
* Perfect for MP/M* Systems
* RAM Kit is very low power (300 MA typical)

32K STATIC RAM KIT — $99.95
For RAM Kit A&T — Add $40

Digital Research Computers

P.O. BOX 381450 • DUNCANVILLE, TX 75138 • (214) 225-2309

TERMS: Add $3.00 postage. Orders under $15 add 75¢ handling. No C.O.D. We accept Visa and MasterCharge. Tex. Res. add 5-1/8% Tax. Foreign orders (except Canada) add 20% P & H. Orders over $50 add 85¢ for insurance.

*TM OF DIGITAL RESEARCH INC. (CALIF.)

WE ARE NOT ASSOCIATED WITH DIGITAL RESEARCH INC. (CALIF.) THE SUPPLIERS OF CPM SOFTWARE

CP/M-68K Conditional Batch Processing

by Roger E. Donais

Anyone who uses batch processing is certain to recognize the value of conditional instructions. A few basic submit statements, such as SKIPIF, PAUSEIF, and ABORTIF have been added to CP/M-80 in the past. These instructions make it possible for compiling, assembling, and linking to be handled by a single submit file that is smart enough to stop the process if any phase should fail.

CP/M-68K and CP/M-80 are similar in many ways. In fact, files and programs can share the same disk without adverse consequences. They are, however, as different as the processors they support. Because CP/M-80 creates a working copy of a submit file, that file becomes the natural target for implementing batch control functions. On the other hand, CP/M-68K stores the expanded file in memory. The fact is that a direct approach would require more knowledge about the undocumented aspects of CP/M-68K than can be acquired in an evening!

IF.68K (Listing One, page 72) began as two outlines. The first listed what was wanted and the other described how it would be accomplished. One pencil later the following compromise was reached:

Syntax: The conditional statement is structured so as to minimize programming effort by taking advantage of the parsing done by the CCP, and yet present a recognizable form of conditional statement.

IF <parameter> <condition> .<action> /$$$$/ <statement>

Condition: The first four characters taken from the file-name field of the file control block created by the CCP are used to identify the condition to be tested. An asterisk may be appended to indicate that the complementary condition is to be used.

- AMBIG tests for an ambiguous parameter (wildcards).
- EXISTS tests for an existing file.
- EMPTY tests for an empty file or nonexistent file.
- NULL tests for an undefined parameter.

A simple "if" can greatly speed the software development process.

Actions: The first character taken from the file-type field of the same file control block is used to identify the desired action. An asterisk may again be appended to indicate that the complementary action is to be taken.

- S(kip) ignores the statement following /$$$$/.
- Q(uit) aborts submit file processing.
- P(ause) prompts for operator assistance.

The double-slash quad-dollar-sign token separates the conditional statement from the executable CP/M com-

Roger E. Donais, 7506 Republic Ct., Alexandria, VA 22306

mand tail. When the submit file is expanded, a dollar sign is used as the leading character to signal formal parameter replacement. If the next character is not an ASCII digit, the character is written to the expanded output and no replacement is performed. This simply means that we use four dollar signs in the submit file but have the program search for only two.

One disadvantage with this compromise approach is that CP/M-68K will pass only one command. This limits the range of the conditional statement to one CP/M-68K command. It's bad enough that the same conditional statement has to be entered over and over again in order to control a range of commands, but the aggravation of watching it reload each time is just too much!

With a little camouflage, CP/M can be tricked into passing more than one command. The best character seems to be a semicolon. Very few programs dare use this character because it marks the beginning of a comment. This makes it a prime candidate for an alternate command separator. It affects very few programs and conceals its true intent from CP/M, and our program need only replace these bogus separators with the exclamation marks that CP/M-68K expects to find.

The problem with this wonderful logic is that CP/M-68K seems to stop building the command tail after it finds a semicolon. Because the basic technique could use any character, a backslash became the next choice. Although the entire command tail is passed, the chain function does not protect the referenced line but leaves it where it is. It doesn't take much imagination to realize what happens when the default buffer is overwritten. So we are limited to using it with built-in commands and the aggravation of using only one external command.

Let's look at the resulting kludge. The program starts by looking for an action character in the second file control block. If it fails to find a match, as would be the case with a null parameter, it flips a switch before testing the first file control block. Control is then transferred according to the indicated action.

Each action begins by making a call to evaluate the condition. This evaluation ends with a jump that complements the original condition according to a NOT condition and a NOT action request. This is definitely a kludge that capitalizes upon the parsing and setup done by the CCP. The file-name and file-type fields are normally padded with spaces (an even \$20). The presence of an asterisk (CP/M wildcard) changes the padding character to a question mark (an odd \$3F). The code uses this even/odd fill character to blindly obtain the desired state.

The two complementary labels, TRUE and FALSE, are truly illusionary. What began as a simple matter

of going to one or the other, based upon the resulting condition, turned into quite a nightmare. At times it seemed that neither one was right. Had they been ZIP and ZAP they might not have confused the issue so much. Debugging eventually reached a point where I would no longer look at them or at the remarks. It became, "Well, it's BEQ now, so let's try BNE" or "target is TRUE, so try FALSE!" It became worse regardless of what I tried. Finally, everything was wrong. That made it easy. TRUE became FALSE and FALSE became TRUE. I no longer care how it looks nor if it makes sense. It works and that is the end of it!

A skip is performed by merely ending the program and returning to the CCP. The command is executed by invoking the CP/M-68K chain function after replacing all semicolons with exclamation marks.

There may be a more direct way to abort than the method used. Programs, however, evolve according to a programmer's knowledge and imagination. My 68000 and CP/M-68K knowledge is very limited, so that left only imagination. The *CP/M User's Guide* says that a submit statement contained in a submit file would transfer control if the referenced file exists. Otherwise the statement would be ignored and processing would resume with the next statement in the current file.

The solution is obvious: invoke an existing, empty submit file—*et voilà*! The actual code goes one step more. It creates a submit file with a single instruction that subsequently erases itself. The result may not be fast, but it does abort without having to keep more junk on the disk.

Yes, the whole thing has the potential to be fooled and foiled. But if you remain within the guidelines and don't wander beyond its incomplete parsing and blind processing, it performs exactly as intended. The crazy little submit file in Listing Two (page 82) may well be the world's slowest ERAQ, but it does provide an example to demonstrate the performance of IF.68K. The additional example in Listing Three (page 82) not only conditionally assembles and links the modules of an assembly project but also copies the changed source files from a RAM disk to floppy. Naturally, execution speed is improved by loading IF.68K from a RAM disk rather than floppy.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 195.**

CP/M-68K (Text begins on page 70)

Listing One

```
*****************************************************************
*                                                               *
* PROGRAM:  IF.68K                  LAST UPDATE:  10 Mar 85     *
*                                                               *
* PURPOSE:  PROVIDE BASIC CONTROL FOR CP/M-68k SUBMIT FACILITY *
*                                                               *
* AUTHOR:   Roger E. Donais         TEL: (703) 765-0615         *
*           7506 Republic Ct.                                   *
*           Alexandria, VA 22306                                *
*                                                               *
* UPDATE LOG:                                                   *
*                                                               *
* 1.2   10 Mar 85 --- Replaced non-working semi-colon with      *
*                         back-slash as command separator and   *
*                         added ^Z check for empty file   ...red *
* 1.1   14 Nov 84 --- Added semi-colon as muliple command       *
*                         separator                       ...red *
* 1.0   11 Jul 84                                               *
*                                                               *
*****************************************************************
*                                                               *
* SYNTAX:                                                       *
*                                                               *
*     IF <Filename> <Condition>.<Action> /$$$$/ <statement>     *
*                                                               *
*     CONDITION IS: AMBIG - TO TEST FOR AMBIGIOUS FILESPEC      *
*                   EXIST - TO TEST FOR AN EXISTING FILE        *
*                   EMPTY - TO TEST FOR AN EMPTY FILE           *
*                   NULL  - TO TEST FOR A NULL PARAMETER        *
*                                                               *
*     ACTION IS:        S - TO SKIP STATMENT PART OF COMMAND LINE *
*                       Q - TO QUIT SUBMIT FILE PROCESS         *
*                       P - TO PAUSE FOR OPERATOR ASSISTANCE    *
*                                                               *
*     STATEMENT IS ANY VALID CP/M COMMAND                       *
*                                                               *
*     NOTE:  AN ASTERISK (*) MAY BE APPENDED TO THE CONDITION   *
*            AND/OR THE ACTION TO INDICATE NEGATION.  AMBIG*    *
*            WOULD THUS TRANSLATE "NOT AMBIGIOUS" AND Q* WOULD  *
*            TRANSLATE "DO NOT QUIT".                           *
```

(Continued on page 74)

Circle no. 89 on reader service card.

Listing One

```
*                                                              *
*        MULTIPLE COMMANDS MAY BE SEPARATED BY SEMI-COLONS, AN *
*        EXCLAIMATION MARK (!) OR END OF LINE ENDS STATEMENT.  *
*                                                              *
****************************************************************

* CP/M FUNCTION DEFINITIONS
bdos            equ  2
fnABORT         equ  0
fnCINP          equ  1
fnCOUT          equ  2
fnPRINT         equ  9
fnRESET         equ 13
fnOPEN          equ 15
fnCLOSE         equ 16
fnDELETE        equ 19
fnREAD          equ 20
fnWRITE         equ 21
fnCREATE        equ 22
fnDMA           equ 26
fnCHAIN         equ 47

.page
* MISC CONSTANTS
FCB1            equ $5C          1st CP/M File Control Block
FCB2            equ $38          2nd CP/M File Control Block
NAM1            equ FCB1+1       1st FCB File Name field
NAM2            equ FCB2+1       2nd FCB File Name field
TYP1            equ NAM1+8       1st FCB File Type field
TYP2            equ NAM2+8       2nd FCB File Type field
BUFF            equ $80          Default CP/M Buffer / Command Line
SIZE            equ $0F+FCB1     CP/M File Size (Sector Count) FOR FCB1

****************************************************************
* PHASE-1 --- PROCESS (P)AUSE), (S)KIP  and (Q)UIT PARAMETERS

        MOVE.L  4(A7),A6         A6 Contains Base Page
        LEA     128(A6),A3       DEFAULT BUFFER
        MOVE.B  TYP2(A6),D0      Get Action Character - 2nd FCB
RETRY:
        CMP.B   #'S',D0
        BEQ     SKIP
        CMP.B   #'Q',D0
        BEQ     QUIT
        CMP.B   #'P',D0
        BEQ     PAUSE
        MOVE.B  TYP1(A6),D0      Get Action Character - 1st FCB
        EOR #1,NULLFLG
        BNE      RETRY
ERROR:
        MOVE.L #MESSAGE,A1
ERROR1:
        MOVE.B (A1)+,D1
        BEQ  ERROR2
        EXT  D1
        MOVE #fnCOUT,D0
        TRAP #bdos
        BRA  ERROR1
ERROR2:
        MOVE #fnABORT,D0
        TRAP #bdos

NULLFLG:
        DC.W 0000

.page
****************************************************************
* PHASE-2 --- PROCESS AMBIG, EXISTS, EMPTY and NULL PARAMETERS
COMPARE:
        MOVE #4-1,d0             4 Character token count
        LEA  NAM2(A6),A0         Get Token address
        TST NULLFLG
```

```
        BEQ L1
        LEA NAM1(A6),A0
L1:     LSL.L #8,D2             Make room for next character
        MOVE.B (A0)+,D2           and load it
        DBRA D0,L1              Repeat for all 4-characters

        MOVE.L #TABLE,A0        Point to Function Table
        MOVE   #4-1,D0          Number of Entries
L2:     MOVE.L (A0)+,D1         Get Current Entry
        CMP.L   D2,D1
        BEQ  JUMP               Exit on Match
        ADDA #4,A0              Else Step to Next Entry
        DBRA D0,L2                and Repeat for ALL Entries
L3:     ADDA #4,A7              Pop callers from stack
        BRA  ERROR                and abort
JUMP:
        TST NULLFLG
        BEQ JMP1                EVERYTHING GOOD ON 2ND FCB
        TST D0
        BNE L3                  AND ONLY NULL ON 1ST FCB
JMP1:   MOVE.L (A0),A0          JUMP TO FUNCTION
        JMP (A0)
.page
******************************************************************
SKIP:   * SKIP COMMAND TAIL IF CONDITION TRUE
        BSR COMPARE
        BNE CHAIN               Execute Command Tail if FALSE
EXIT:   RTS                     and Ignore if TRUE

******************************************************************
QUIT:   * QUIT (ABORT) SUBMIT IF CONDITION TRUE
        BSR COMPARE
        BNE CHAIN               Ignore Command Tail if FALSE
ABORT:
        MOVE.L  #ABORTMSG,D1
        MOVE    #fnPRINT,D0
        TRAP    #bdos
```

(Continued on next page)

CP/M-68K (Listing continued, text begins on page 70)
Listing One

```
        MOVE      #fnRESET,DØ
        TRAP      #bdos           Make Certain Drive A: is R/W
        MOVE.L    #ABORTREC,D1
        MOVE      #fnDMA,DØ       Setup for File Output
        TRAP      #bdos
        MOVE.L    #ABORTFCB,D1
        MOVE      #fnDELETE,DØ    Delete any '$$$ABORT.SUB' File
        TRAP      #bdos
        MOVE      #fnCREATE,DØ    Then make a new one
        TRAP      #bdos
        MOVE      #fnWRITE,DØ     Fill it
        TRAP      #bdos
        MOVE      #fnCLOSE,DØ     Close it
        TRAP      #bdos
        MOVE.L    #ABORTSUB,D1    and Chain to it
        BRA       CHAIN1

*****************************************************************
PAUSE:  * PAUSE IF CONDITION TRUE
        BSR COMPARE
        BNE CHAIN                 EXECUTE W/O STOPPING IF FALSE
   X9:
        MOVE.L #PROMPT,D1
        MOVE #fnPRINT,DØ
        TRAP #bdos
        MOVE #fnCINP,DØ
```

(Continued on page 78)

Listing One

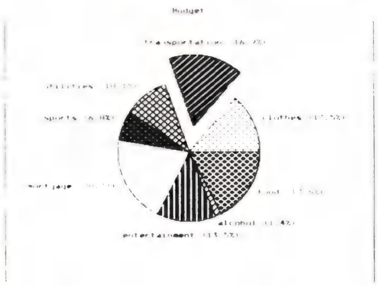
```
        TRAP #bdos
        AND  #$5F,D0              Force upper case
        CMP.B   #'Q',D0
        BEQ     ABORT             Quit on "Q"
        CMP.B   #'S',D0
        BEQ     EXIT              Skip on "S"
        CMP.B   #'C',D0
        BNE     X9                And continue on "C"
.page
CHAIN:  * CHAIN TO COMMAND TAIL

        ADD     #BUFF,A6          Point to command line
        MOVE.B  (A6)+,D0          Get command length
        EXT     D0

X5:     DBRA D0,X6                Ignore Statement if Command
        RTS                         Tail cannot be found

X6:     CMP.B #'/',(A6)+
        BNE X5                    Look for leading slash

        CMP.B #' ',-2(A6)         Insure that the previous
        BEQ X7                      character was either
        CMP.B #9,-2(A6)             a space or tab
        BNE X5
X7:
        CMP.B #'$',(A6)+          The Original four dollar
        BNE X5                      signs will appear
        CMP.B #'$',(A6)+            as only two, so
        BNE X5                      check for 2 $'s


        CMP.B #'/',(A6)+          Check for trailing slash
        BNE X5

        CMP.B #' ',(A6)+          And finally delimiting
        BEQ X8                      space or tab
        CMP.B #9,-1(A6)
        BNE X5
X8:
* So we got the command tail. Let's take a quick break
* and repalce all semicolons with exclaimation marks.
        move.l A6,A1              copy pointer and
        move   D0,D1              character count,
        bra Y10                   then begin search
Y9:
          move.b #'!',-1(A1)
Y10:
            cmp.b #'\',(A1)+
          dbeq D1,Y10             search until eol or '\'
        beq Y9                    substitute & continue if '\'

        SUB #1,A6                 Step back to new len position
        ADD #1,D0                   adjust remaining command length
        MOVE.B D0,(A6)              and punch it
        MOVE.L A6,D1              Put the command tail in D1
CHAIN1:
        MOVE #fnDMA,D0            Make addr in D1
        TRAP #bdos                the Default Buffer
        MOVE #fnCHAIN,D0
        TRAP #bdos                and Chain it

***************************************************************
OPEN:
        MOVE.L  A6,D1             Base page to D1
        ADD.L   #FCB1,D1            and step to fcb
        MOVE    #fnOPEN,D0
        TRAP #bdos
        CMP.B   #$FF,D0           Set flags on open file
        RTS
.page
NULL1:  CMP.B #$20,NAM1(A6)       Test for no file name
        BNE TRUE
```

(Continued on page 80)

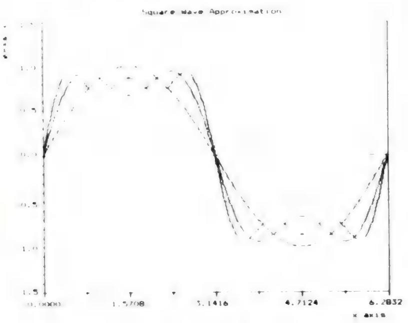
GRAF 3.0

the complete
BUSINESS and SCIENTIFIC
printer graphics program

$49.95 CP/M-80 | $69.95 MS-DOS / PC-DOS



- display floating point data directly from spreadsheets, data bases, word processors, and programming languages (or the keyboard) in a wide variety of bar, pie, line, and scatter plots
- plot and group up to 6 different variables on a single graph, distinguished by up to 14 different fill-in patterns and 8 different point-plotting symbols
- menu driven operation supports automatic graph scaling, labeling, and legend creation
- add up to 5 different-density grid lines, and choose from a wide variety of numerical labeling options



GRAF 2.0 Update Policy: Returning your original GRAF 2.0 disk to MSC entitles you to $20.00 off the above prices.
Numeric Compressor Option: 8087 support for the MS-DOS/PC-DOS version of GRAF 3.0 is available for an additional $20.00.

TERMS: We ship via first class mail. The above prices include domestic shipping and handling. Orders outside USA require additional $5.00 for postage. N.Y. residents add state sales tax. When ordering you MUST state your computer and printer make and model. We support MS-DOS (PC-DOS) version 2.0 or later on computers with at least 192k RAM, and CP/M-80 version 2.2 or later on Z80 computers (other than modified Apples) supporting a TPA of at least 54k (requires 64k of RAM). Most soft-sector disk formats are available. (If you can read several formats, please send us a list.) GRAF 3.0 works with any printer fully compatible with one of the following: Epson FX, RX, LX, MX (with GRAFTRAX), or LQ-1500; C. Itoh Prowriter; NEC 8023A; Star Micronics Gemini 10X, 15X, SG-10, SG-15; IBM Graphics Printer; Okidata 192, and earlier Okidata models equipped with the "IBM Plug 'n' Play" chips. (If you have an Okidata printer, other than the 192, the Plug 'n' Play chips are *required*!)

MSC Microcomputer Systems Consultants

27 Forest Avenue
Port Jefferson Station
New York 11776-1820

Circle **no. 136** on reader service card.

BD Software, Inc., maker of the original CP/M-80 C Language Development System, knows

Time is precious

So the compilation, linkage and execution speeds of BDS C are the fastest available, even (especially!) on floppy-based systems. Just ask any user! With 15,000+ packages sold since 1979, there are *lots* of users . . .

New! Ed Ream's RED text editor has been integrated into the package, making BDS C a truly complete, self-contained C development system.

Powerful original features: CDB symbolic source-level debugger, fully customizable library and run-time package (for convenient ROM-ing of code), XMODEM-compatible telecommunications package, and other sample applications.

National C User's Group provides direct access to the wealth of public-domain software written in BDS C, including text editors and formatters, BBS's, assemblers, C compliers, games and much more.

Complete package price: $150.
All soft-sectored disk formats, plus Apple CP/M, available off-the-shelf. Shipping: free, by UPS, within USA for *prepaid* orders. Canada: $5. Other: $25. VISA, MC, COD, rush orders accepted.

BDSoftware, Inc.

BD Software, Inc.
P O Box 2368
Cambridge MA 02238
617 • 576 • 3828

Circle **no. 123** on reader service card.

Listing One

```
FALSE:   MOVE #0,D0                Start with FALSE
         BRA TOGGLE

TRUE:    MOVE #1,D0                Start with TRUE
TOGGLE:  EOR.B D0,TYP2-1(A6)       TOGGLE "NOT CONDITION"
         MOVE.B TYP2-1(A6),D0
         EOR.B D0,TYP2+2(A6)       TOGGLE "NOT ACTION"
         BTST #0,TYP2+2(A6)        FLAG RESULT
         RTS

**************************************************************
NULL:    * TEST FIRST FCB FOR NULL PARAMETER

         TST NULLFLG               If FCB2 is in use the
         BEQ NULL1                 Condition is FCB1 Filename
         EOR.B  D0,TYP1-1(A6)      else TOGGLE
         MOVE.B TYP1-1(A6),D0      using FCB1 Pad Characters
         EOR.B  D0,TYP1+2(A6)
         BTST #0,TYP1+2(A6)
         RTS

**************************************************************

EMPTY:   * TEST FIRST FCB FOR NULL, NON-EXISTANT OR EMPTY FILE

         CMP.B #20,NAM1(A6)        NO FILE NAME IS EMPTY FILE
         BEQ FALSE
         BSR OPEN                  NON-EXISTANT IS ALSO EMPTY
         BEQ FALSE
         MOVEQ #fnREAD,D0
         TRAP  #2
         TST   D0
```

```
        BNE FALSE                   NO RECORD == EMPTY FILE
        CMP.B #$1A,(A3)
        BEQ FALSE                   ^Z MUST BE EMPTY ASCII FILE
        BRA TRUE

*************************************************************
EXIST:   * TEST FIRST FCB FOR EXISTING FILE

        BSR OPEN
        BNE FALSE                   File not Found Therefore Non-Existant
        BRA TRUE                    otherwise it's naturally there

*************************************************************
AMBIG:   * TEST FIRST FCB FOR AMBIGIOUS NAME OR TYPE

        MOVE #11-1,DØ               Test 8-Char NAME + 3-Char TYPE
    X4:
          CMP.B #'?',NAM1(A6,DØ)
        DBEQ DØ,X4
        BNE TRUE                    Contains "?" Therefore Ambigious
        BRA FALSE
.page
*************************************************************
******************** DATA STORAGE AREA **********************
TABLE   DC.B 'AMBI'
        DC.L AMBIG              ***********************************
        DC.B 'EMPT'             *              NOTE               *
        DC.L EMPTY              *                                 *
        DC.B 'EXIS'             * NULL MUST BE THE LAST TABLE     *
        DC.L EXIST              * ENTRY. SEE COMPARE SUBROUTINE   *
        DC.B 'NULL'             *                                 *
        DC.L NULL               ***********************************

ABORTSUB:
        DC.B 1Ø,'A:$$$ABORT',Ø
ABORTREC:
        DC.B 'ERA A:$$$$$$ABORT.SUB',$D,$A
```

(Continued on next page)

Listing One

```
        DC.L ';SUBMIT FILE PROCESSING ABORTED',$D,$A,$1A
ABORTFCB:
        DC.B 1,'$$$ABORTSUB'
        DC.B 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
ABORTMSG
        DC.B $D,$A,'ABORTING ...$'

PROMPT:
  DC.B $7,$D,$A,$D,$A,'SUBMIT FILE PROCESSING PAUSED.',$D,$A,$D,$A

  DC.B $7,'    PRESS: (Q) TO QUIT, (S) TO SKIP  -or-  (C) TO CONTINUE $'

MESSAGE:
  DC.B 7,$D,$A,$A
  DC.B 'IF.68K Ver 1.2 CONDITIONAL SUBMIT COMMAND FORMAT:',$D,$A,$A
  DC.B '  IF <filespec>  AMBIGx.yx /$$$$/ <command tail>',$D,$A
  DC.B '  IF <filespec>  EXISTx.yx /$$$$/ <command tail>',$D,$A
  DC.B '  IF <filespec>  EMPTYx.yx /$$$$/ <command tail>',$D,$A
  DC.B '  IF <parameter>  NULLx.yx /$$$$/ <command tail>',$D,$A,$A

  DC.B '  WHERE: x is an optional "*" to indicate negation',$D,$A
  DC.B '     and y is (Q)uit, (S)kip or (P)ause',$D,$A,$A

  DC.B '  NOTE:  Command tail may consist of multiple commands',$D,$A
  DC.B '         separated by backslashes (\) and ends at an',$D,$A
  DC.B '         exclamation mark (!) or physical end of line.',$D,$A,$A

  DC.B '  EXAMPLE:  IF A:JUNK.TYP  /$$$$/ EMPTY.S CMD1\CMD2!CMD3',$D,$A,$A
  DC.B '    Neither CMD1 nor CMD2 will execute if A:JUNK.TYP',$D,$A
  DC.B '    is empty.  Regardless, CMD3 will always execute.'

  DC.B $D,$A,$A,0

        END
```

End Listing One

Listing Two

```
              ERAQ.SUB  -  File Deletion with Query

DIR $1
;-------------------------------------------------------------
IF $1  EXISTS.P /$$$$/ ERA $1
IF $2  NULL.Q   /$$$$/
SUBMIT ERAQ $2 $3 $4 $5 $6 $7 $8 $9
```

End Listing Two

Listing Three

```
             ASM.SUB  -  Simple 68K Assemble and Link

IF $1 NULL.S /$$$$/ ERA $1.O
IF FILE1.BAK EXISTS.S* /$$$$/ ERA FILE1.O\PIP B:=FILE1.S
IF FILE2.BAK EXISTS.S* /$$$$/ ERA FILE2.O\PIP B:=FILE2.S
IF FILE3.BAK EXISTS.S* /$$$$/ ERA FILE3.O\PIP B:=FILE3.S
IF FILE4.BAK EXISTS.S* /$$$$/ ERA FILE4.O\PIP B:=FILE4.S
ERA *.BAK
IF FILE1.O   EXISTS.S  /$$$$/ A:AS68 -S A: -L FILE1.S
IF FILE2.O   EXISTS.S  /$$$$/ A:AS68 -S A: -L FILE1.S
IF FILE3.O   EXISTS.S  /$$$$/ A:AS68 -S A: -L FILE1.S
IF FILE4.O   EXISTS.S  /$$$$/ A:AS68 -S A: -L FILE1.S
A:LO68 -R FILE1.O FILE2.O FILE3.O FILE4.O
```

End Listings

FORTRAN PROGRAMMERS

Discover why
you should be using
F77L
the complete implementation
of the ANSI FORTRAN 77
Standard for the IBM PC and
compatibles.
If you are serious about your
FORTRAN programming, you
should be using F77L.
$477

Lahey Computer Systems, Inc.
31244 Palos Verdes Drive West, Suite 243
Rancho Palos Verdes, California 90274
(213) 541-1200
Serving the FORTRAN community since 1969

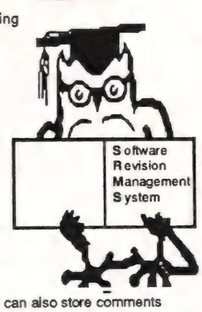
Circle **no. 67** on reader service card.

TOTAL RECALL

FOR $104.95

Now you can retrieve everything you entered during program development -- from version one thru all updates. SRMS does it with a complete source utility at an affordable price.

You can retrieve specific versions of a program, make changes and reinstate the source while recording when, why and where changes were made.



All versions are stored in a single library without duplication of common code or text -- saves disk space. You can also store comments specific to each version -- provides complete development history and documentation of the program.

SRMS supports recall and edit of programs written in BASIC, FORTRAN, PASCAL (including Turbo Pascal), C and ASSEMBLY CODE.

Requires MS or PC DOS, Version 2 and up with 128K and one disk (floppy or hard).

Quilt Computing 7048 Stratford Road
Woodbury, MN 55125
(612) 739-4650

Circle **no. 107** on reader service card.

Finally. BSW-Make.

The Boston Software Works now brings a complete implementation of the Unix "make" facility to MS-DOS. No more recompiling every file in sight after a small edit; no more wondering if you've really rebuilt every module affected by an edit. Just type "make" and BSW-Make automatically builds your product quickly, efficiently and correctly.

BSW-Make supports:

- most compilers and assemblers
- MS-DOS or PC-DOS v2.00 or later
- macros for parameterized builds
- default rules
- MS-DOS pathnames
- any MS-DOS machine (192K minimum)

Only $69.95 postpaid (Mass. residents add 5% sales tax)

The Boston Software Works
120 Fulton Street, Boston, MA 02109
(617) 367-6846

Circle **no. 126** on reader service card.

LEARN PROLOG

Interactively and Quickly with PROLOG-86

Ideal for Artificial Intelligence and Prototyping

A complete *Prolog Interpreter, Tutorial,* and set of *Sample Programs:*

☐ **Modify and write Expert Systems.**
Use the simple "Guess the animal" example on the Tutorial or use the sophisticated system for Section 318 of the US Tax Code written by one of the PROLOG-86 authors and published in the March, 1985 issue of Dr. Dobb's Journal.

☐ **Understand Natural Language**
Use the sample program that produces a dBase DISPLAY command as output.

☐ **Write Symbolic Math or Abstract Problem Solving Applications**
This is a complete Prolog program to convert from Farenheit to Centigrade: f_to_c(C,F):- C is(F-32) *5/9. Planning programs and games are included to help you learn.

☐ **BECOME FAMILIAR WITH PROLOG IN ONE EVENING.**

Programming experience is not required, but a logical mind is. Serious development of experimental systems is practical with PROLOG-86. 1 or 2 pages in Prolog is often equivalent to 10 or 15 in C.

RECENT IMPROVEMENTS: MSDOS commands, on-line help, load Editor.

AVAILABILITY: All MSDOS, PCDOS systems.

ONLY
$125
Full refund if not satisfied during first 30 days.

Solution Systems™

335-D Washington St.
Norwell, Mass. 02061
617-659-1571
800-821-2492

Circle **no. 148** on reader service card.

LEARN LISP

Interactively and Write Realistic Programs with LISP-86 for Only $75

A "COMMON LISP" compatible Tutorial, Interpreter, Debugging, and Pretty Printer plus a Fast, Full Screen Editor, Samples and Help

☐ **Start Easily and Quickly:**
A complete, modular tutorial helps you learn LISP at your own pace. An integrated, interactive environment provides all of the elements needed to enter, modify, analyze and debug programs.

☐ **Natural Language, Expert Systems and Mailing List:**
Natural Language concepts are illustrated by a phone number retrieval program. Choose the best word processing program for you with the Expert System. File handling and typical data processing work are demonstrated by a Mailing List program.

☐ **Write Realistic Programs:**
Short examples and substantial programs of about 10 pages in length help you learn by modifying, studying and using the key concepts needed to write programs of 1000 lines or more.

☐ **The "COMMON LISP" Standard:**
LISP-86 includes a 230+ function subset of the "COMMON LISP" Standard. Use extras like the MSDOS interface and graphics. Or use "strict compatibility" to make programs written in LISP-86, with no changes, work with other COMMON LISP systems like VAX LISP, GC/LISP or LISP Machine LISP.

Use and Modify the "Which Word Processor?" program to learn about EXPERT SYSTEMS.

Runs on any MSDOS or PCDOS System: Not copy-protected, LISP-86 is available in just about any 3", 5" or 8" format. PC compatibles can run LISP-86 with no installation procedure. 192K memory and 1 floppy drive are the minimums required.

For Beginners and Experienced Programmers

ONLY
$75
Full refund if not satisfied during first 30 days.

Solution Systems™

335-B Washington St.
Norwell, Mass. 02061
617-659-1571
800-821-2492

Circle **no. 149** on reader service card.

ProDOS and the RAM Disk

by Alfred Steele

I have a Synetix Flashcard with 288K of RAM that I use with Apple's CP/M, Pascal, and DOS 3.3. When I got my card more than two years ago, there was no ProDOS. After using ProDOS several times, I knew that I could not live without my RAM disk, so I decided to roll my own driver. I did several things to speed up the process of writing the driver. I used only 256K of the 288K memory on the Flashcard. The Flashcard uses six banks of memory. Four banks have 64K and the other two have only 16K each. I chose to use only the four 64K banks of memory to save development time and reduce code size. The loss of 32K of storage space is not a problem at this time. With the 256K of RAM in use, I get a RAM disk with 512 blocks of storage space.

Using a RAM disk speeds hi-res display by a factor of four.

This is Version 1.3 of the RAM-disk driver (see Listing, page 85). Version 1.0 was larger and faster but not relocatable. It had to be loaded in page 3 of memory. Version 1.1 fixed bugs in the installation program. Version 1.2 is the first version inich the RAM-disk driver code is small enough to fit inside of a free space in ProDOS. The program is completely relocatable anywhere in memory. One side effect of making the driver smaller and relocatable is that it is slower than in the other versions.

Version 1.3 fixes some errors in the installation code and adds more user prompts. This version of the RAM-disk driver can be installed anywhere in memory, but only three options are given at run time. You may place the driver inside a free space in ProDOS at $FF00. This space may only be free in ProDOS Version 1.01. Later versions of ProDOS may not support the driver at that address. You may also place the driver under the BASIC interpreter or at $300 hex page 3. The reason I advise you to install the driver inside of ProDOS is to support MousePaint, which uses page 3 of memory. Information on how to use the interpreter can be found in the *ProDOS Technical Reference Manual* and *Beneath Apple ProDOS*. Both books are really necessary.

The code for the RAM-disk driver is not hard to follow. The RAM-disk driver code that will be relocated to any address starts at label SSDSTART and ends at ENDSSD. The routine is only 8E hex (142 decimal) bytes long. The main part of the installation code starts at label MAIN. This is a series of subroutines to check if the RAM disk has already been formatted. If this is a first time installation, the program title and the volume name of the RAM disk are displayed. The driver code is relocated to the address of your choice. The ProDOS active devices are listed and the count is updated. The address of the device driver is placed in the vector table for Slot 5, Drive 1. The RAM-disk volume directory information is written to the RAM disk and the volume bit map is written to disk. Information on the diskette volume can be found in Chapter 4 of *Beneath Apple ProDOS* and Appendix B of the *ProDOS Technical*

Alfred Steele, Box 6296, APO NY 09012

Reference Manual.

The program has been written to be easy to use. If you try to install the driver more than once, the program will display a message that a driver has already been installed and abort. If the driver finds a formatted volume directory header on the RAM disk it will ask you if you want to clear it or not. If you type N, the program will install the driver but won't clear the RAM-disk directory. If you type Y, the program will install the driver and reformat the RAM disk. If any other driver, such as a hard disk driver, is already installed in the active device vector for Slot 5, Drive 1, the program will abort with an error message.

Once the RAM-disk driver is installed you can access the RAM disk using standard ProDOS commands and utilities. You can catalog the RAM disk using either the command CAT /RAM or CAT,S5,D1. If you want to copy files to the RAM disk use your copy of ProDOS FILER [*or see the article "Adding a COPY Command to ProDOS" on page 54 of this issue—ed*]. The system intergradation is clean and efficient and provides super fast loading and storing of any type of file.

One benchmark that I ran using the RAM disk consisted of loading high-resolution pictures to the screen. I made up a ProDOS test disk with a short BASIC program to load 16 different hi-res pictures as fast as possible. Using Apple's Disk II it took about 35 seconds to display all 16 pictures. Using the same program and pictures loaded to the RAM disk it only took 8.3 seconds to display all 16 pictures. The program runs about 4.2 times faster using the RAM disk for an effective I/O transfer speed of 15.9K per second vs. the Disk II transfer speed of 3.8K per second.

For those who don't want to type in more than 900 lines of code and comments, I will send you a copy of the source and object code on one Apple ProDOS disk for $15 postpaid anywhere in the United States. Please send a money order because a personal check is difficult to process in West Germany. To assemble the code, you need a copy of the ProDOS assembler/editor, or you can assemble the code under a DOS 3.3 assember and move the object code to a ProDOS disk. My address is:

Alfred Steele
Box 6296
APO NY 09012

DDJ

Ram Disk Listing (Text begins on page 84)

```
SOURCE    FILE #01 =>/RAMDISK.SOURCE/SSD3
0000:             **********************************
0000:             *                                *
0000:             *    FLASHCARD RAMDISK DRIVER    *
0000:             *                                *
0000:             *          VERSION 1.3           *
0000:             *           JULY 1985            *
0000:             *                                *
0000:             *         COPYRIGHT 1985         *
0000:             *         ALFRED STEELE          *
0000:             *                                *
0000:             * THIS PROGRAM MAY BE COPIED     *
0000:             * FOR PERSONAL, NON-PROFIT USE   *
0000:             *                                *
0000:             **********************************
0000:             *

0000:                        LST   ON
----- NEXT OBJECT FILE NAME IS /RAMDISK.SOURCE/SSD3.0
2000:       2000             ORG   $2000
2000:             *
2000:             * TEXT STUFF
2000:             *
2000:       0000  EOL        EQU   $00
2000:       008D  CR         EQU   $8D
2000:             *
2000:             * ADDRESS OF SSD DRIVER
2000:             *
2000:       0000  TEMPZERO   EQU   $00
2000:       0002  SSDCODE    EQU   $02
2000:             *
2000:             * MLI DISK DRIVE ZERO PAGE
2000:             *
2000:       0001  READ       EQU   $01
2000:       0002  WRITE      EQU   $02
2000:       0006  DIRBLOCKS  EQU   $06
2000:       0008  SAVEBYTES  EQU   $08
2000:             *
2000:             * DEVICE DRIVER PARAMETERS
2000:             *
2000:       0042  COMMAND    EQU   $42
2000:       0043  UNITNUMB   EQU   $43
2000:       0044  BUFFADR    EQU   $44
2000:       0046  BLOCKNUM   EQU   $46
2000:       0048  RAMBANK    EQU   $48
2000:       004A  TEMP       EQU   $4A
2000:             *
2000:             * LOW BYTE ADDRESS OF BLOCK
2000:             * NUMBERS IN SSD RAM DISK.
2000:             *
2000:       00D2  BLOCK1     EQU   $D2
2000:       00D3  BLOCK2     EQU   $D3
2000:       00D4  BLOCK3     EQU   $D4
2000:       00D5  BLOCK4     EQU   $D5
2000:       00D6  BLOCK5     EQU   $D6
2000:       00D7  BLOCK6     EQU   $D7
2000:             *
2000:             * SSD ADDRESS FOR CARD IN SLOT 5
2000:             *
2000:       C0D0  SSDLOW     EQU   $C0D0
2000:       C0D1  SSDHI      EQU   $C0D1
2000:             *
2000:       BEF5  GETBUFF    EQU   $BEF5
2000:       BF00  MLI        EQU   $BF00
2000:       BF31  DEVCNT     EQU   $BF31
2000:       BF32  DEVLST     EQU   $BF32
2000:       BF1A  DEVADR51   EQU   $BF1A
2000:             *
2000:             * VALUE OF DEVICE DRIVER POINTER
2000:             * IF THE SLOT AND DRIVE IS NOT
2000:             * BEING USED.
2000:             *
2000:       D0A2  NODEV      EQU   $D0A2
2000:             *
2000:             * BI GLOBAL PAGE ADDRESS
2000:             *
2000:       BE00  WARMDOS    EQU   $BE00
2000:             *
2000:             * HARDWARE LOCATIONS
2000:             *
2000:       C000  KEY        EQU   $C000
2000:       C010  STROBE     EQU   $C010
2000:             *
2000:             * MONITOR SUBROUTINES
2000:             *
2000:       FC58  HOME       EQU   $FC58
2000:       FD8E  CROUT      EQU   $FD8E
2000:       FDED  COUT       EQU   $FDED
2000:       FF3A  BELL       EQU   $FF3A
2000:             *
2000:             **********************************
2000:             *
2000:4C 90 21                JMP   MAIN
2003:             *
2003:             **********************************
2003:             *
2003:             * THIS IS THE SSD RAMDISK DRIVER
2003:             * CODE THAT IS MOVED TO SSDCODE
2003:             * AT THE START OF THIS PROGRAM.
2003:             * THIS ROUTINE MUST START ON A
2003:             * PAGE BOUNDARY. I USE THE DUMMY
2003:             * DS STATEMENT BELOW SO THAT THE
2003:             * ROUTINE WILL START ON A PAGE
2003:             * BOUNDARY. IF YOU CHANGE ANY
2003:             * CODE ABOVE BE SURE TO ADJUST
2003:             * THE DS STATEMENT BEFORE YOU
2003:             * ASSEMBLE AND RUN THE PROGRAM.
2003:             *
2003:             *
2003:       00FD             DS    $FD
2100:             *
2100:             *
2100:D8           SSDSTART   CLD
2101:             *
2101:             * SAVE ALL MLI ZERO PAGE LOC
2101:             * PLUS TWO NEEDED BY DRIVER
2101:             *
2101:AD D3 C0                LDA   $C0D3
2104:A2 08                   LDX   #SAVEBYTES
2106:B5 42        S1         LDA   COMMAND,X
2108:48                      PHA
2109:CA                      DEX
210A:10 FA  2106             BPL   S1
210C:             *
210C:             * COMPUTE WHICH 64K BLOCK WE
210C:             * NEED TO READ OR WRITE TO.
210C:             * EACH 64K BLOCK OF RAM HAS 128
210C:             * PRODOS 512 BYTES BLOCKS IN IT.
210C:             * WE ARE ONLY USING THE 64K
210C:             * BANKS SO ONLY BLOCKS:
210C:             *
210C:             *     BLOCK1    BLOCKS   0-127
210C:             *     BLOCK2    BLOCKS 128-255
210C:             *     BLOCK4    BLOCKS 256-383
210C:             *     BLOCK5    BLOCKS 384-511
```

(Continued on next page)

Ram Disk Listing

(Listing continued, text begins on page 84)

```
210C:                  *
210C:                  * ARE BEING USED. WHEN WE KNOW
210C:                  * WHAT BANK WE WILL USE WE MOVE
210C:                  * THAT BANKS LOW BYTE FOR THE
210C:                  * SSD RAM CARD BANK ADDRESS INTO
210C:                  * THE LOW-BYTE OF THE ZERO PAGE
210C:                  * POINTER RAMBANK. THE HI-BYTE
210C:                  * IS SET TO #$C0. IF BLOCKNUM+1
210C:                  * IS NOT ZERO WE NEED TO ACCESS
210C:                  * BLOCKS 256-383 OR 384-511.
210C:                  *
210C:A5 47             COMPUTE   LDA   BLOCKNUM+1
210E:D0 0C     211C              BNE   C2
2110:                  *
2110:                  * BLOCKS 0-127 OR 128-255
2110:                  *
2110:A5 46                       LDA   BLOCKNUM
2112:30 04     2118              BMI   C1
2114:                  *
2114:                  * BLOCKS 0-127
2114:                  *
2114:A9 D2                       LDA   #BLOCK1
2116:D0 0E     2126              BNE   C4
2118:                  *
2118:                  * BLOCKS 128-255
2118:                  *
2118:A9 D3             C1        LDA   #BLOCK2
211A:D0 0A     2126              BNE   C4
211C:                  *
211C:                  * BLOCKS 256-383 OR 384-511
211C:                  *
211C:A5 46             C2        LDA   BLOCKNUM
211E:30 04     2124              BMI   C3
2120:                  *
2120:                  * BLOCKS 256-383
2120:                  *
2120:A9 D5                       LDA   #BLOCK4
2122:D0 02     2126              BNE   C4
2124:                  *
2124:                  * BLOCKS 384-511
2124:                  *
2124:A9 D6             C3        LDA   #BLOCK5
2126:85 48             C4        STA   RAMBANK
2128:                  *
2128:                  * SET UP HI-BYTE OF SSD CARD
2128:                  *
2128:A9 C0                       LDA   #$C0
212A:85 49                       STA   RAMBANK+1
212C:                  *
212C:                  * DECODE TYPE OF COMMAND FOR
212C:                  * RAM DISK DRIVER TO DO.
212C:                  *
212C:A5 42                       LDA   COMMAND
212E:85 4A                       STA   TEMP
2130:F0 48     217A              BEQ   EXIT
2132:                  *
2132:                  * SAVE COMMAND IN Y-REG AND
2132:                  * MASK OUT BITS 1 AND 0.
2132:                  * IF A-REG = ZERO AFTER MASK
2132:                  * THEN THIS IS A VALID READ
2132:                  * OR WRITE COMMAND.
2132:                  *
2132:29 FC                       AND   #$FC
2134:F0 06     213C              BEQ   SETBLOCK
2136:                  *
2136:                  * ERROR COMMAND HANDLER
2136:                  * INVALID COMMAND SENT
2136:                  *
2136:A9 27                       LDA   #$27
2138:85 4A                       STA   TEMP
213A:D0 3E     217A              BNE   EXIT
213C:                  *
213C:                  * READ OR WRITE COMMAND. SET UP
213C:                  * BLOCK NUMBER AND BRANCH TO
213C:                  * READBYTE OR WRITEBYTE.
213C:                  *
213C:                  * EACH 64K BANK HOLDS 128 PRODOS
213C:                  * 512 BYTE BLOCKS. GET THE LOWER
213C:                  * 7 BITS AND SHIFT ONCE LEFT.
213C:                  * THIS WILL MULTPLY THE NUMBER
213C:                  * 0-127 TO 0-254 EVEN VALUES
213C:                  * ONLY. THIS ADJUSTED BLOCK
213C:                  * NUMBER IS USED AS THE HI-BYTE
213C:                  * OF THE ADDRESS IN THE 64K BANK
213C:                  * THE LOW-BYTE IS SET TO ZERO
213C:                  * AND INC FROM 0 TO $FF.
213C:                  *
213C:A5 46             SETBLOCK  LDA   BLOCKNUM
213E:29 7F                       AND   #$7F
2140:0A                          ASL
2141:85 47                       STA   BLOCKNUM+1
2143:8D D1 C0                    STA   SSDHI
2146:                  *
2146:                  * MOVE COMMAND TO A-REG AND
2146:                  * SET UP LOOP COUNT TO RUN
2146:                  * THRU READ OR WRITE TWICE.
2146:                  * EACH LOOP THRU READ OR WRITE
2146:                  * WILL MOVE 256 BYTES.
2146:                  *
2146:A0 00                       LDY   #0
2148:84 46                       STY   BLOCKNUM
```

```
214A:A2 02                        LDX    #2
214C:                   *
214C:                   * READ OR WRITE ONE 512 BYTE
214C:                   * PRODOS BLOCK.
214C:                   *
214C:8C D0 C0           SSDIO     STY    SSDLOW
214F:84 4A                        STY    TEMP
2151:                   *
2151:                   * CHECK A-REG TO SEE IF WE NEED
2151:                   * TO DO A READ OR WRITE.
2151:                   *
2151:                   *   A-REG = $01 FOR READ
2151:                   *   A-REG = $02 FOR WRITE
2151:                   *
2151:                   * ROTATE A-REG THRU CARRY AND
2151:                   * CHECK IF CARRY SET OR CLEAR.
2151:                   * IF CARRY SET THEN READ. IF
2151:                   * CARRY CLEAR THEN WRITE.
2151:                   *
2151:A5 42                        LDA    COMMAND
2153:4A                           LSR
2154:90 0B      2161              BCC    WRITEBYTE
2156:                   *
2156:A0 00              READBYTE  LDY    #0
2158:B1 48                        LDA    (RAMBANK),Y
215A:A4 4A                        LDY    TEMP
215C:91 44                        STA    (BUFFADR),Y
215E:18                           CLC
215F:90 08      2169              BCC    CONT
2161:                   *
2161:                   * WRITE BYTE TO RAMDISK
2161:                   *
2161:B1 44              WRITEBYTE LDA    (BUFFADR),Y
2163:A0 00                        LDY    #0
2165:91 48                        STA    (RAMBANK),Y
2167:A4 4A                        LDY    TEMP
2169:                   *
2169:                   * COMMON CODE TO INC TO NEXT
2169:                   * BYTE TO READ OR WRITE
2169:                   *
2169:C8                 CONT      INY
216A:D0 E0      214C              BNE    SSDIO
216C:                   *
216C:E6 45                        INC    BUFFADR+1
216E:E6 47                        INC    BLOCKNUM+1
2170:A5 47                        LDA    BLOCKNUM+1
2172:8D D1 C0                     STA    SSDHI
2175:CA                           DEX
2176:D0 D4      214C              BNE    SSDIO
2178:                   *
2178:84 4A                        STY    TEMP
217A:                   *
217A:                   * THIS PART OF THE ROUTINE IS
217A:                   * ENTERED IF THE STATUS, READ
217A:                   * OR WRITE I/O IS COMPLETE.
217A:                   *
217A:                   * SSD EXIT RETURN
217A:                   *
217A:A4 4A              EXIT      LDY    TEMP
217C:A2 00                        LDX    #0
217E:68                 EX1       PLA
217F:95 42                        STA    COMMAND,X
2181:E8                           INX
2182:E0 09                        CPX    #SAVEBYTES+1
2184:D0 F8      217E              BNE    EX1
2186:                   *
2186:AD D2 C0                     LDA    $C0D2
2189:18                           CLC
218A:98                           TYA
218B:F0 01      218E              BEQ    EX2
218D:38                           SEC
218E:                   *
218E:60                 EX2       RTS
218F:EA                 ENDSSD    NOP
2190:                   *
2190:                   ********************************
2190:                   *
2190:                   * THIS IS THE MAIN SET OF JSR
2190:                   * STATEMENTS TO INSTALL AND
2190:                   * INIT THE RAMDISK.
2190:                   *
2190:20 50 22           MAIN      JSR    CHECKDEV
2193:20 97 23                     JSR    WHERETO
2196:20 4F 25                     JSR    TITLE
2199:20 64 25                     JSR    DISPVOL
219C:                   *
219C:20 B4 21                     JSR    RELOCATE
219F:20 C1 21                     JSR    SETDEV
21A2:20 D7 21                     JSR    SETVECTOR
21A5:                   *
21A5:                   * CHECK CLEAR DIRECTORY FLAG
21A5:                   * IF CLEARDIR = 1 THEN SKIP
21A5:                   * THE ROUTINES TO WRITE A NEW
21A5:                   * DIRECTORY TO THE RAMDISK AND
21A5:                   * SKIP THE ROUTINE TO WRITE A
21A5:                   * NEW BITMAP TO THE RAMDISK.
21A5:                   *
21A5:AD 4A 26                     LDA    CLEARDIR
21A8:D0 09      21B3              BNE    MN1
21AA:                   *
21AA:                   * DO A NORMAL FIRST TIME INIT
21AA:                   *
21AA:20 E2 21                     JSR    CLEAREND
21AD:20 EF 21                     JSR    WRITEDIR
21B0:20 35 22                     JSR    BITMAP
21B3:60                 MN1       RTS
21B4:                   *
21B4:                   ********************************
21B4:                   *
21B4:                   * THIS ROUTINE IS USED TO MOVE
21B4:                   * THE SSD RAM DISK DRIVER TO
21B4:                   * THE ADDRESS OF SSDCODE.
21B4:                   * THE ROUTINE IS COMPLETELY
21B4:                   * RELOCATABLE AND DOES NOT HAVE
21B4:                   * TO BE ON A PAGE BOUNDARY.
21B4:                   *
21B4:A0 00              RELOCATE  LDY    #0
21B6:B9 00 21           R1        LDA    SSDSTART,Y
21B9:91 02                        STA    (SSDCODE),Y
21BB:C8                           INY
21BC:C0 8F                        CPY    #ENDSSD-SSDSTART
21BE:D0 F6      21B6              BNE    R1
21C0:60                           RTS
21C1:                   *
```

(Continued on next page)

Ram Disk Listing (Listing continued, text begins on page 84)

```
21C1:                ********************************
21C1:                *
21C1:                * THIS ROUTINE IS USED TO MAKE
21C1:                * ROOM IN THE DEVICE LIST OF
21C1:                * ACTIVE DEVICES FOR THE RAM
21C1:                * DISK DRIVE IDENTIFICATION.
21C1:                *
21C1:AE 31 BF        SETDEV    LDX   DEVCNT
21C4:E8                        INX
21C5:8E 31 BF                  STX   DEVCNT
21C8:                *
21C8:BD 31 BF        SD1       LDA   DEVLST-1,X
21CB:9D 32 BF                  STA   DEVLST,X
21CE:CA                        DEX
21CF:D0 F7    21C8             BNE   SD1
21D1:                *
21D1:                * STORE THE RAM DISK IN LIST
21D1:                *
21D1:A9 50                     LDA   #$50
21D3:8D 32 BF                  STA   DEVLST
21D6:60                        RTS
21D7:                *
21D7:                ********************************
21D7:                *
21D7:                * INSTALL RAMDISK DRIVER VECTOR
21D7:                *
21D7:A5 02           SETVECTOR LDA   SSDCODE
21D9:8D 1A BF                  STA   DEVADR51
21DC:A5 03                     LDA   SSDCODE+1
21DE:8D 1B BF                  STA   DEVADR51+1
21E1:60                        RTS
21E2:                *
21E2:                ********************************
21E2:                *
21E2:                * CLEAR BUFFER FROM END OF DISK
21E2:                * DATA TO END OF DISK BUFFER.
21E2:                *
21E2:A0 00           CLEAREND  LDY   #0
21E4:98                        TYA
21E5:99 82 26        CE        STA   MIDBUFF,Y
21E8:99 82 27                  STA   MIDBUFF+$100,Y
21EB:C8                        INY
21EC:D0 F7    21E5             BNE   CE
21EE:60                        RTS
21EF:                *
21EF:                ********************************
21EF:                *
21EF:                * CALL MLI TO WRITE OUT THE
21EF:                * SSD RAM DISK VOLUME DIRECTORY
21EF:                *
21EF:A9 06           WRITEDIR  LDA   #DIRBLOCKS
21F1:8D 7E 26                  STA   BUFFER+39
21F4:A9 02                     LDA   #$02
21F6:8D 4F 26                  STA   WRITEBLOCK
21F9:                *
21F9:20 00 BF                  JSR   MLI
21FC:81                        DB    $81
21FD:4B 26                     DW    MLIWRITE
21FF:                *
21FF:                * CLEAR AND SET BLOCKS
21FF:                * 3 THRU DIRBLOCKS OF
21FF:                * VOLUME DIRECTORY. WE HAVE TO
21FF:                * SET UP THE LAST/NEXT BLOCK
21FF:                * NUMBERS IN THE FIRST FOUR (4)
21FF:                * BYTES OF EACH BLOCK.
21FF:                *
21FF:A2 03                     LDX   #3
2201:20 28 22        CB1       JSR   CLEARBUFF
2204:                *
2204:CA                        DEX
2205:8E 57 26                  STX   BUFFER
2208:E8                        INX
2209:E8                        INX
220A:                *
220A:E0 06                     CPX   #DIRBLOCKS
220C:D0 07    2215             BNE   CB2
220E:                *
220E:A9 00                     LDA   #0
2210:8D 59 26                  STA   BUFFER+2
2213:F0 03    2218             BEQ   CB3
2215:                *
2215:8E 59 26        CB2       STX   BUFFER+2
2218:                *
2218:                * WRITE VOLUME DIRECTORY BLOCK
2218:                * TO SSD RAM DISK.
2218:                *
2218:CA              CB3       DEX
2219:8E 4F 26                  STX   WRITEBLOCK
221C:                *
221C:20 00 BF                  JSR   MLI
221F:81                        DB    $81
2220:4B 26                     DW    MLIWRITE
2222:                *
2222:E8                        INX
2223:E0 06                     CPX   #DIRBLOCKS
2225:D0 DA    2201             BNE   CB1
2227:60                        RTS
2228:                *
2228:                ********************************
2228:                *
2228:                * CLEAR ENTIRE FILE BUFFER
2228:                *
```

```
2228:A0 00           CLEARBUFF LDY   #0
222A:98                        TYA
222B:99 57 26        CL1       STA   BUFFER,Y
222E:99 57 27                  STA   BUFFER+$100,Y
2231:C8                        INY
2232:D0 F7    222B             BNE   CL1
2234:60                        RTS
2235:                *
2235:                ******************************
2235:                *
2235:                * SET ALL BLOCKS NOT USED IN
2235:                * RAM DISK TO HEX $FF TO SHOW
2235:                * THAT THOSE BLOCKS ARE FREE.
2235:                * SET FIRST BYTE TO SHOW THAT
2235:                * BLOCKS 0 THRU DIRBLOCKS ARE
2235:                * IN USE.
2235:                *
2235:A9 01           BITMAP    LDA   #$01
2237:8D 57 26                  STA   BUFFER
223A:A0 3F                     LDY   #63
223C:A9 FF                     LDA   #$FF
223E:                *
223E:99 57 26        BM2       STA   BUFFER,Y
2241:88                        DEY
2242:D0 FA    223E             BNE   BM2
2244:                *
2244:A9 06                     LDA   #DIRBLOCKS
2246:8D 4F 26                  STA   WRITEBLOCK
2249:                *
2249:                * WRITE OUT BIT MAP TO SSD
2249:                * RAM DISK.
2249:                *
2249:20 00 BF                  JSR   MLI
224C:81                        DB    $81
224D:4B 26                     DW    MLIWRITE
224F:                *
224F:60                        RTS
2250:                *
2250:                ******************************
2250:                *
2250:                * THIS ROUTINE IS USED TO CHECK
2250:                * FOR A DEVICE DRIVER ALREADY
2250:                * INSTALLED FOR SLOT AND DRIVE
2250:                * WE WANT TO USE. I USE SLOT 5
2250:                * DRIVE 1 FOR THE RAMDISK DRIVER
2250:                * IF THE ADDRESS AT SLOT 5 DRIVE
2250:                * 1 DOES NOT POINT TO $D0A2 THE
2250:                * PROGRAM WILL ABORT WITH AN
2250:                * ERROR MESSAGE. IF WE FIND NO
2250:                * DRIVER INSTALL I READ BLOCK 2
2250:                * OF THE RAMDISK AND COMPARE IT
2250:                * WITH THE VOLUME DIRECTORY
2250:                * HEADER INFORMATION AT BUFFER.
2250:                * IF THE INFORMATION IS THE SAME
2250:                * A MESSAGE IS PRINTED ASKING
2250:                * THE USER IF HE WANTS TO ZERO
2250:                * THE DIRECTORY ON THE RAMDISK
2250:                * OR NOT. IF WE WANT TO CLEAR
2250:                * THE DIRECTORY I SET CLEARDIR
2250:                * TO ZERO. IF WE DON'T WANT TO
2250:                * CLEAR THE DIRECTORY I SET
2250:                * CLEARDIR TO ONE.
2250:                *
2250:AD 1A BF       CHECKDEV  LDA   DEVADR51
2253:C9 A2                     CMP   #>NODEV
2255:D0 7A    22D1             BNE   DEVABORT
2257:                *
2257:AD 1B BF                  LDA   DEVADR51+1
225A:C9 D0                     CMP   #<NODEV
225C:D0 73    22D1             BNE   DEVABORT
225E:                *
225E:                * LETS TRY AND READ THE RAM DISK
225E:                * DIRECTORY AT BLOCK 2 AND
225E:                * COMP IT TO SOME OF THE BYTES
225E:                * AT ADDR BUFFER TO SEE IF THE
225E:                * RAMDISK MIGHT HAVE A FORMATTED
225E:                * DIRECTORY ON IT.
225E:                *
225E:A9 01                     LDA   #READ
2260:85 42                     STA   COMMAND
2262:                *
2262:                * ADDRESS OF READ BUFFER
2262:                *
2262:A9 82                     LDA   #>MIDBUFF
2264:85 44                     STA   BUFFADR
2266:A9 26                     LDA   #<MIDBUFF
2268:85 45                     STA   BUFFADR+1
226A:                *
226A:                * BLOCK NUMBER TO READ
226A:                *
226A:A9 02                     LDA   #$02
226C:85 46                     STA   BLOCKNUM
226E:A9 00                     LDA   #$00
2270:85 47                     STA   BLOCKNUM+1
2272:                *
2272:                * READ THE RAMDISK USING DRIVER
2272:                * AT SSDSTART IN THIS PROGRAM
2272:                *
2272:20 00 21                  JSR   SSDSTART
2275:                *
2275:                * WE WILL NOW ASSUME AT THERE
2275:                * IS THE INFORMATION IN BLOCK 2
2275:                * OF THE RAMDISK AT ADDRESS
2275:                * MIDBUFF. COMPARE THREE SETS OF
2275:                * BYTES TO SEE IF IT MATCHS A
2275:                * VOLUMN DIRECTORY HEADER BLOCK.
2275:                * IF ANY BYTE DOES NOT MATCH
2275:                * THE ROUTINE JUMPS TO NODIR
2275:                * IF ALL THREE SETS OF BYTES
2275:                * MATCH I ASK USER IF HE WANTS
2275:                * TO CLEAR THE DIR OR NOT.
2275:                *
2275:A2 00                     LDX   #$00
2277:BD 57 26        CD0       LDA   BUFFER,X
227A:DD 82 26                  CMP   MIDBUFF,X
227D:D0 4C    22CB             BNE   NODIR
```

(Continued on next page)

```
227F:                   *
227F:E8                          INX
2280:E0 04                       CPX    #$04
2282:D0 F3     2277              BNE    CD0
2284:                   *
2284:A2 14                       LDX    #$14
2286:BD 57 26           CD1      LDA    BUFFER,X
2289:DD 82 26                    CMP    MIDBUFF,X
228C:D0 3D     22CB              BNE    NODIR
228E:                   *
228E:E8                          INX
228F:E0 25                       CPX    #$25
2291:D0 F3     2286              BNE    CD1
2293:                   *
2293:A2 27                       LDX    #$27
2295:BD 57 26           CD2      LDA    BUFFER,X
2298:DD 82 26                    CMP    MIDBUFF,X
229B:D0 2E     22CB              BNE    NODIR
229D:                   *
229D:E8                          INX
229E:E0 2B                       CPX    #$2B
22A0:D0 F3     2295              BNE    CD2
22A2:                   *
22A2:                   * ASK USER IF HE WANT TO ZERO
22A2:                   * DIRECTORY OR NOT. IF THE USER
22A2:                   * WANTS TO ZERO THE DIRECTORY
22A2:                   * JUST CONTINUE WITH THE INSTALL
22A2:                   * PROGRAM. IF THE USER DOES NOT
22A2:                   * WANT TO ZERO THE DIRECTORY
22A2:                   * DO A JUMP TO THE DOS WARMSTART
22A2:                   *
22A2:20 58 FC                    JSR    HOME
22A5:20 3A FF                    JSR    BELL
22A8:                   *
22A8:A0 00                       LDY    #0
22AA:B9 EF 22           CD3      LDA    DIRFOUND,Y
22AD:C9 00                       CMP    #EOL
22AF:F0 09     22BA              BEQ    CD4
22B1:                   *
22B1:09 80                       ORA    #$80
22B3:20 ED FD                    JSR    COUT
22B6:C8                          INY
22B7:4C AA 22                    JMP    CD3
22BA:                   *
22BA:20 41 25           CD4      JSR    GETKEY
22BD:C9 59                       CMP    #'Y'
22BF:F0 0A     22CB              BEQ    NODIR
22C1:                   *
22C1:C9 4E                       CMP    #'N'
22C3:D0 F5     22BA              BNE    CD4
22C5:                   *
22C5:                   * THE USER DOES NOT WANT TO
22C5:                   * CLEAR THE DIRECTORY.
22C5:                   *
22C5:A9 01                       LDA    #$01
22C7:8D 4A 26                    STA    CLEARDIR
22CA:60                          RTS
22CB:                   *
22CB:                   * WE WANT TO CLEAR THE DIRECTORY
22CB:                   * ON THE RAMDISK SO SET FLAG FOR
22CB:                   * CLEAR AND RETURN TO ROUTINE.
22CB:                   *
22CB:A9 00              NODIR    LDA    #$00
22CD:8D 4A 26                    STA    CLEARDIR
22D0:60                          RTS
22D1:                   *
22D1:                   * THIS PART OF THE ROUTINE IS
22D1:                   * ENTERED IF WE FIND A DEVICE
22D1:                   * DRIVER ADDRESS THAT DOES NOT
22D1:                   * POINT TO THE NO DEVICE DRIVER
22D1:                   * ACTIVE ADDRESS OR TO THE
22D1:                   * ADDRESS WHERE WE WANT TO PUT
22D1:                   * OUR CODE. THERE COULD BE A
22D1:                   * HARD DISK OR OTHER STORAGE
22D1:                   * DEVICE ALREADY INSTALL FOR
22D1:                   * THAT SLOT/DRIVE.
22D1:                   *
22D1:20 58 FC           DEVABORT JSR    HOME
22D4:20 3A FF                    JSR    BELL
22D7:                   *
22D7:A0 00                       LDY    #0
22D9:B9 4B 23           DA1      LDA    ABORTMSG,Y
22DC:C9 00                       CMP    #EOL
22DE:F0 09     22E9              BEQ    DA2
22E0:                   *
22E0:09 80                       ORA    #$80
22E2:20 ED FD                    JSR    COUT
22E5:C8                          INY
22E6:4C D9 22                    JMP    DA1
22E9:                   *
22E9:20 3A FF           DA2      JSR    BELL
22EC:4C 00 BE                    JMP    WARMDOS
22EF:                   *
22EF:                   *******************************
22EF:                   *
22EF:                   * DIRECTORY FOUND MESSAGE
22EF:                   *
22EF:8D 8D 8D 8D        DIRFOUND DFB    CR,CR,CR,CR
22F3:8D 8D 8D 8D                 DFB    CR,CR,CR,CR
22F7:20 20 20 20                 ASC    '
2303:44 49 52 45                 ASC    'DIRECTORY      FOUND'
2312:8D 8D                       DFB    CR,CR
2314:20 20 20 20                 ASC    '                                  '
```

(Continued on page 92)

Ram Disk Listing (Listing continued, text begins on page 84)

```
231A:43 4C 45 41              ASC    'CLEAR          DIRECTORY "Y" OR "N"'
2334:8D 8D                    DFB    CR,CR
2336:20 20 20 20              ASC    '
234A:00                       DFB    EOL
234B:               *
234B:               * ABORT MESSAGE
234B:               *
234B:8D 8D 8D 8D    ABORTMSG  DFB    CR,CR,CR,CR
234F:8D 8D 8D 8D              DFB    CR,CR,CR,CR
2353:20 20 20 20              ASC    '
2357:44 45 56 49              ASC    'DEVICE         DRIVER ALREADY INSTALLED'
2376:8D 8D                    DFB    CR,CR
2378:20 20 20 20              ASC    '
2385:50 52 4F 47              ASC    'PROGRAM        ABORT'
2392:8D 8D 8D 8D              DFB    CR,CR,CR,CR
2396:00                       DFB    EOL
2397:               *
2397:               ********************************
2397:               *
2397:               * THIS ROUTINE IS USED TO ASK
2397:               * THE USER WHERE HE/SHE WOULD
2397:               * LIKE TO MOVE THE RAMDISK
2397:               * DRIVER CODE TO. THERE ARE
2397:               * THREE LOCATIONS TO MOVE THE
2397:               * CODE TO OR THE USER CAN QUIT
2397:               *
2397:               * LOC 1: INSIDE PRODOS AT $F000
2397:               * LOC 2: BETWEEN THE BASIC
2397:               *        INTERPRETER AND THE
2397:               *        FILE BUFFERS.
2397:               * LOC 3: AT $300
2397:               *
2397:20 58 FC       WHERETO:  JSR    HOME
239A:A0 00                    LDY    #0
239C:B9 0F 24       WT0       LDA    WHEREMSG,Y
239F:C9 00                    CMP    #EOL
23A1:F0 09    23AC            BEQ    WT1
23A3:               *
23A3:09 80                    ORA    #$80
23A5:20 ED FD                 JSR    COUT
23A8:C8                       INY
23A9:4C 9C 23                 JMP    WT0
23AC:               *
23AC:20 3A FF       WT1       JSR    BELL
23AF:20 41 25                 JSR    GETKEY
23B2:C9 31                    CMP    #'1'
23B4:F0 15    23CB            BEQ    WT2
23B6:               *
23B6:C9 32                    CMP    #'2'
23B8:F0 20    23DA            BEQ    WT3
23BA:               *
23BA:C9 33                    CMP    #'3'
23BC:F0 2A    23E8            BEQ    WT4
23BE:               *
```

(Continued on page 94)

```
23BE:C9 51              CMP   #'Q'
23C0:D0 EA     23AC     BNE   WT1
23C2:              *
23C2:20 58 FC           JSR   HOME
23C5:20 3A FF           JSR   BELL
23C8:4C 00 BE           JMP   WARMDOS
23CB:              *
23CB:              * MOVE CODE INSIDE PRODOS
23CB:              *
23CB:A9 00         WT2      LDA   #$00
23CD:85 02              STA   SSDCODE
23CF:A9 FF              LDA   #$FF
23D1:85 03              STA   SSDCODE+1
23D3:AD 81 C0           LDA   $C081
23D6:AD 81 C0           LDA   $C081
23D9:60                 RTS
23DA:              *
23DA:              * MOVE CODE UNDER BI
23DA:              *
23DA:A9 01         WT3      LDA   #1
23DC:20 F5 BE           JSR   GETBUFF
23DF:B0 10     23F1     BCS   BUFFERROR
23E1:              *
23E1:85 03              STA   SSDCODE+1
23E3:A9 00              LDA   #$00
23E5:85 02              STA   SSDCODE
23E7:60                 RTS
23E8:              *
23E8:              * MOVE CODE TO $300
23E8:              *
23E8:A9 00         WT4      LDA   #$00
23EA:85 02              STA   SSDCODE
23EC:A9 03              LDA   #$03
23EE:85 03              STA   SSDCODE+1
23F0:60                 RTS
23F1:              *
23F1:              * ERROR GETTING BUFFER FROM SYS
23F1:              *
23F1:20 3A FF      BUFFERROR JSR   BELL
23F4:20 58 FC           JSR   HOME
23F7:A0 00              LDY   #0
23F9:B9 B3 24      BE0      LDA   NOBUFF,Y
23FC:C9 00              CMP   #EOL
23FE:F0 09     2409     BEQ   BE1
2400:              *
2400:09 80              ORA   #$80
2402:20 ED FD           JSR   COUT
2405:C8                 INY
2406:4C F9 23           JMP   BE0
2409:              *
2409:20 3A FF      BE1      JSR   BELL
240C:4C 00 BE           JMP   WARMDOS
240F:              *
240F:              ********************************
```

(Continued on page 96)

Circle no. 84 on reader service card.

UNIX/XENIX COMPATIBILITY!

C• lib

C FUNCTION LIBRARY

C-LIB incorporates over 200 routines to extend the capabilities of C on the IBM PC. Versions are available for the Lattice, Microsoft, and DeSmet (C Ware) C compilers. All libraries are compatible with PC-DOS 2.0+ operating systems.

Featuring:

- UNIX-Compatible "CURSES" Screen/Window Processing Library
- Buffered, Interrupt-driven, Asynchronous Communications Library
- Powerful String Functions
- 8087/Microsoft Floating Point Conversion

C-LIB SOFTWARE AND MANUAL PACKAGE $195.

Please call or write for additional information.
DEMO DISKETTE AVAILABLE, please include $5.

VANCE info systems

2818 clay street • san francisco, ca 94115 • (415) 922-6539

IBM & PC-DOS are trademarks of International Business Machines Corp. Lattice is a trademark of Lattice Inc. Xenix & Microsoft C are trademarks of Microsoft Inc. UNIX is a trademark of Bell Labs. C-LIB is a trademark of VANCE info systems.

Circle no. 154 on reader service card.

SERIOUS SOFTWARE
FOR
SERIOUS DEVELOPERS

FABS PLUS

(B-Tree Structure) Rapid access and maintenance of very large files with millions of records. Extremely fast on unlimited number of keys. Index utility provides protection from corrupted index files. Can be used on files as large as your system can hold.

RETAIL PRICE $195.00

Network versions of FABS PLUS also available for PC-DOS 3.XX and Novell Netware.

RETAIL PRICE $395.00

AUTOSORT

Sort/Merge subsystem for very large files. In stand-alone or callable sub-routine, extremely fast. Sorts files as large as the operating system and workspace will allow.

RETAIL PRICE $150.00

Versions available for FABS and AUTOSORT:
CPM-80, CP/M-86, MP/M-11, MS-DOS, PC-DOS, MICROSOFT BASIC(S), COBOL, FORTRAN, PASCAL, PL/1, CBASIC, CB80, CB86, CBASIC86, LATTICE C

DB-FABS

Data, Screen, Report Manager: Can be used in stand-alone and run-time modes. Creates files and forms for the management and access of data. Report and screen handler to generate listings. B-Tree indexing and high speed sort capabilities.

RETAIL PRICE $295.00

Version of DB-FABS available for Novell Netware

RETAIL PRICE $395.00

COMPUTER CONTROL SYSTEMS, INC.
ROUTE 3, BOX 168, LAKE CITY, FL 32055 (904) 752-0912

Circle no. 127 on reader service card.

```
240F:                 *
240F:                 * MESSAGE ASKING USER WHERE
240F:                 * TO PUT RAMDISK DRIVER CODE.
240F:                 *
240F:8D 8D 8D 8D      WHEREMSG  DFB   CR,CR,CR,CR
2413:20 20 20 20                ASC   '
2418:52 41 4D 44                ASC   'RAMDISK        DRIVER ADDRESS
242F:4F 50 54 49                ASC   'OPTIONS'
2436:8D 8D 8D                   DFB   CR,CR,CR
2439:20 20 20 20                ASC   '
243F:5B 31 5D 20                ASC   '[1]            INSIDE PRODOS AT $FF00'
2459:8D 8D                      DFB   CR,CR
245B:20 20 20 20                ASC   '
2461:5B 32 5D 20                ASC   '[2]            UNDER BASIC INTERPRETER'
247C:8D 8D                      DFB   CR,CR
247E:20 20 20 20                ASC   '
2484:5B 33 5D 20                ASC   '[3]            AT $300
248F:8D 8D                      DFB   CR,CR
2491:20 20 20 20                ASC   '
2497:5B 51 5D 55                ASC   '[Q]UIT'
249D:8D 8D                      DFB   CR,CR
249F:20 20 20 20                ASC   '
24A9:20 20 20 20                ASC   '
24B2:00                         DFB   EOL
24B3:                 *
24B3:                 ********************************
24B3:                 *
24B3:                 * NO DOS BUFFER FOUND UNDER BI
24B3:                 *
24B3:8D 8D 8D 8D      NOBUFF    DFB   CR,CR,CR,CR
24B7:8D 8D 8D 8D                DFB   CR,CR,CR,CR
24BB:20 20 20 20                ASC   '
24C9:46 41 54 41                ASC   'FATAL          ERROR'
24D4:8D 8D                      DFB   CR,CR
24D6:20 20 20                   ASC   '               '
24D9:43 4F 55 4C                ASC   'COULD          NOT ALLOCATE BUFFER BETWEEN
'
24FA:8D 8D                      DFB   CR,CR
24FC:20 20 20 20                ASC   '                    '
2506:42 49 20 41                ASC   'BI             AND FILE BUFFERS'
2519:8D 8D                      DFB   CR,CR
251B:20 20 20 20                ASC   '               '
2520:46 4C 41 53                ASC   'FLASHCARD      DRIVER NOT INSTALLED'
253E:8D 8D                      DFB   CR,CR
2540:00                         DFB   EOL
2541:                 *
2541:                 ********************************
2541:                 *
2541:                 * THIS ROUTINE IS USED TO GET
2541:                 * A SINGLE KEYPRESS FROM THE
2541:                 * KEYBOARD AND RETURN IT IN
2541:                 * THE A-REG WITH THE MSB OFF.
2541:                 *
2541:8D 10 C0         GETKEY    STA   STROBE
2544:AD 00 C0         GK1       LDA   KEY
2547:10 FB    2544              BPL   GK1
2549:8D 10 C0                   STA   STROBE
254C:29 7F                      AND   #$7F
254E:60                         RTS
254F:                 *
254F:                 ********************************
254F:                 *
254F:                 * PRINT PROGRAM TITLE
254F:                 *
254F:20 58 FC         TITLE     JSR   HOME
2552:A0 00                      LDY   #0
2554:B9 9B 25         TI1       LDA   MSG,Y
2557:C9 00                      CMP   #EOL
2559:F0 08    2563              BEQ   TI2
255B:                 *
255B:09 80                      ORA   #$80
255D:20 ED FD                   JSR   COUT
2560:C8                         INY
2561:D0 F1    2554              BNE   TI1
2563:60               TI2       RTS
2564:                 *
2564:                 ********************************
2564:                 *
2564:                 * THIS ROUTINE WILL DISPLAY THE
2564:                 * RAM DISK VOLUMN NAME IF THE
```

(Continued on page 98)

PCTEX Now for PC Users:

Professional Typesetting Capability

PCTEX brings to the personal computer user the ability to put any kind of information on paper in a professional, elegant manner. It brings the full power and flexibility of TEX implementations on mainframes to owners of IBM PC's, AT's and workalike computers.

PCTEX is widely used for formatting technical and mathematical material. It is also perfectly suited for producing professional-quality reports, manuals, even books.

PCTEX offers a wide range of typefaces, and a wide choice of drivers which output the finished material on dot matrix printers (Epson, Toshiba), low-cost laser printers (Apple LaserWriter, Corona LP300) and graphics screen preview (Hercules, EGA). All text in this ad has been formatted by PCTEX and produced on a Corona Laser Printer.

Join hundreds of satisfied PCTEX users. Write or call us today.

PCTEX: only $279. Dot-matrix drivers: $100. Laser drivers: $300. Preview (Hercules GC): $250. MF Medley (44 fonts, including Computer Helvetica): $100. Corona Laser Printer and PCTEX: complete $3395. System requirements: DOS 2.0 or later, 512K RAM, 10M hard disk. M/C, Visa accepted.

Personal TEX Inc

20 Sunnyside, Suite H
Mill Valley, CA 94941
(415) 388-8853 Telex 275611

Trademarks: PCTEX, Personal TEX, Inc.; TEX, American Mathematical Society; IBM PC and AT, IBM Corp; LaserWriter, Apple Computer, Inc.; Hercules Graphics Card, Hercules Computer Technology.

Circle no. 76 on reader service card.

THE BEST Z80 ASSEMBLER ON THE MARKET JUST GOT BETTER!

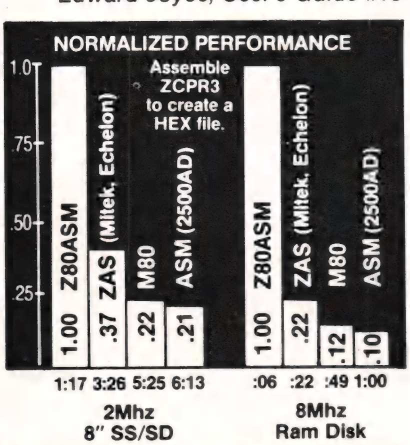
Z80ASM

NOW ONLY $49.95

DON'T ASK HOW OURS CAN BE SO FAST . . . ASK WHY THEIRS ARE SO SLOW!

". . a breath of fresh air . ."
Computer Language, Feb. 85

". . in two words, I'd say speed & flexibility",
Edward Joyce, User's Guide #15



Now fully compatible with M80 in .Z80 mode with many extensions. Time & date in listing, 16 char. externals, plus many other features.

To order, or to find out more about our complete family of development tools, call or write:

SLR Systems

1622 N. Main St., Butler, PA 16001
(800) 833-3061, (412) 282-0864
Telex 559215 SLR SYS

C.O.D., Check or Money Order Accepted

SHIPPING: USA/CANADA + $3 • OTHER AREAS + $10
Z80 CP/M compatibility required.

Circle no. 78 on reader service card.

$49.95 FMT $49.95

Text Formatting System

FMT provides most of the features of the high-priced Text Formatters at our inexpensive price -- and it's easier to use, too! Note the features below:

- Easily configured to your printer. Configuration files for 20 + printer models are provided or generate your own.
- FMT gets the most from your printers by taking advantage of their special features, including condensed, double width, enhanced, double print, italics, elite, letter quality, multiple fonts, etc.
- Multiple modes and combinations of modes can be used on the same line or even in the same word.
- FMT works with your favorite editor!
- FMT uses meaningful mneumonic commands in the style of SCRIPT or ROFF (each command appears on its own input line), including commands for the various printing modes.
- No embedded control codes - you don't have to remember those strange escape/control sequences.
- FMT runs at the maximum speed your printer allows for each printing mode - graphics mode is not required.
- Standard formatting features provided, including headers and footers, automatic page numbering, text justification, tabs for table generation, and embedded files up to TEN deep.
- FMT automatically builds Table of Contents, List of Figures, and three level alphabetized Index.
- Detailed 100 + page manual profusely illustrated with examples.
- Works equally well with IBM-PC, TI-PC, IBM clones and look-alikes (PC-DOS/MS-DOS 128k). Also works with CP/M 8080 and Z80 systems with 64k.
- $49.95 plus $2.00 shipping and handling.

Specify system.

VISA and Master Card Accepted

Dealer Inquiries Welcome

TINY TEK, INC.

Route 1, Box 795
Quinlan, Texas 75474
(214) 447-3025

Circle no. 119 on reader service card.

Ram Disk Listing (Listing continued, text begins on page 84)

```
2564:                * INSTALL WAS SUCCESSFULL.
2564:                *
2564:A0 00           DISPVOL   LDY    #0
2566:B9 31 26        DV0       LDA    VOLNAME,Y
2569:C9 00                     CMP    #EOL
256B:F0 09    2576             BEQ    DV1
256D:09 80                     ORA    #$80
256F:20 ED FD                  JSR    COUT
2572:C8                        INY
2573:4C 66 25                  JMP    DV0
2576:                *
2576:A0 00           DV1       LDY    #0
2578:B9 5C 26        DV2       LDA    VOLUME,Y
257B:C9 00                     CMP    #EOL
257D:F0 0C    258B             BEQ    DV3
257F:C0 0E                     CPY    #14
2581:F0 08    258B             BEQ    DV3
2583:                *
2583:09 80                     ORA    #$80
2585:20 ED FD                  JSR    COUT
2588:C8                        INY
2589:D0 ED    2578             BNE    DV2
258B:                *
258B:20 8E FD        DV3       JSR    CROUT
258E:20 8E FD                  JSR    CROUT
2591:20 8E FD                  JSR    CROUT
2594:20 8E FD                  JSR    CROUT
2597:20 3A FF                  JSR    BELL
259A:60                        RTS
259B:                *
259B:                ********************************
259B:                *
259B:                * PROGRAM TITLE
259B:                *
259B:8D 8D 8D        MSC       DFB    CR,CR,CR
259E:8D 8D 8D                  DFB    CR,CR,CR
25A1:20 20 20 20               ASC    '                       '
25AA:50 52 4F 44               ASC    'PRODOS       RAMDISK DRIVER'
25BF:8D 8D                     DFB    CR,CR
25C1:20 20 20 20               ASC    '                       '
25CE:46 4F 52 20               ASC    'FOR          FLASHCARD'
25DB:8D 8D                     DFB    CR,CR
25DD:20 20 20 20               ASC    '                       '
25EB:56 45 52 53               ASC    'VERSION      1.3'
25F6:8D 8D                     DFB    CR,CR
25F8:20 20 20 20               ASC    '                       '
2605:41 4C 46 52               ASC    'ALFRED       STEELE'
2612:8D 8D                     DFB    CR,CR
2614:20 20 20 20               ASC    '                       '
2623:4A 55 4C 59               ASC    'JULY         1985'
262C:8D 8D 8D 8D               DFB    CR,CR,CR,CR
2630:00                        DFB    EOL
2631:                *
2631:                *
2631:                *
2631:20 20 20 52     VOLNAME   ASC    '                 RAMDISK VOLUME NAME /'
2649:00                        DFB    EOL
264A:                *
264A:                ********************************
264A:                *
264A:                * CLEAR DIRECTORY FLAG
264A:                * SET FLAG FOR NO CLEAR
264A:                *
264A:01              CLEARDIR  DB     $01
264B:                *
264B:                ********************************
264B:                *
264B:                * MLI WRITE PARAMETER LIST
264B:                *
264B:03              MLIWRITE  DB     $03
264C:50                        DB     $50
264D:57 26                     DW     BUFFER
264F:02 00           WRITEBLOCK DW    $0002
2651:                *
2651:                ********************************
2651:                *
2651:                * MLI READ PARAMETER LIST
2651:                *
2651:03              MLIREAD   DB     $03
2652:50                        DB     $50
2653:00 20           READBUFF  DW     $2000
2655:02 00           READBLOCK DW     $0002
2657:                *
2657:                ********************************
2657:                *
2657:                * MLI READ BUFFER
2657:                *
2657:                * THE BYTES FROM BUFFER TO
2657:                * MIDBUFF IS THE RAMDISK VOLUME
2657:                * DIRECTORY HEADER. THESE BYTES
2657:                * HAVE TO BE WRITTEN TO THE
2657:                * RAMDISK AT BLOCK 2 BEFORE THE
2657:                * RAMDISK CAN BE USED BY PRODOS
2657:                *
2657:                * WE WILL ALSO USE PARTS OF
2657:                * THIS VOLUME DIRECTORY HEADER
2657:                * AS A CHECK TO SEE IF THE
2657:                * RAMDISK IS ALREADY FORMATTED
2657:                * IF SO THE USER CAN REFORMAT
2657:                * THE RAMDISK OR LEAVE IT ALONE
2657:                *
2657:00 00           BUFFER    DB     $00,$00
2659:                * LAST BLOCK
2659:03 00                     DB     $03,$00
265B:                * STORAGE TYPE/NAME LENGTH
265B:F3                        DB     $F3
265C:                * VOLUME NAME
265C:52 41 4D        VOLUME    ASC    'RAM'
265F:00 00 00 00               DB     $00,$00,$00,$00
2663:00 00 00 00               DB     $00,$00,$00,$00
2667:00 00 00 00               DB     $00,$00,$00,$00
266B:                * RESERVED
266B:00 00 00 00               DB     $00,$00,$00,$00
266F:00 00 00 00               DB     $00,$00,$00,$00
2673:                * CREATION DATE/TIME
2673:00 00 00 00               DB     $00,$00,$00,$00
2677:                * VERSION/MIN VERSION
2677:00 00                     DB     $00,$00
2679:                * ACCESS BYTE
2679:43                        DB     $43
267A:                * ENTERY LENGTH
267A:27                        DB     $27
267B:                * ENTERIES PER BLOCK
267B:0D                        DB     $0D
267C:                * FILE COUNT ON DISK
267C:00 00                     DW     $0000
267E:                * BLOCK NUMBER OF BIT MAP
267E:06 00                     DW     $0006
2680:                * TOTAL NUMBER BLOCKS ON DISK
2680:00 02                     DW     $0200
```

(Continued on page 100)

Introducing Periscope II Professional Debugger and Break-out Switch

New Periscope II includes a remote break-out switch that does not need its own slot!

(Periscope is) "the best value in development tools currently on the market . . . the most essential element of my 'developer's toolbox'." —Jeff Garbers

"Very powerful for debugging and testing . . . Better than Atron by far." —Wynn Bailey

The break-out switch ***"really sets Periscope apart from the typical software-only debuggers."*** Hung system or locked keyboard? Press the switch to get control!

Periscope's symbol support ***"beats the daylights out of snooping through a map file and making notes".*** See high-level line numbers and source code, too!

"Feel right at home" **in no time** with commands that logically extend Debug's!

Periscope's speed makes other debuggers ***"look absolutely sluggish"!*** It's written entirely in assembler and uses DOS only for file access.

Has all the standard features plus:

- Debug with over 75 breakpoint options

New! Write your own breakpoint tests
New! Traceback
New! Do in-line symbolic assembly

- Debug using one or two monitors
- Recall command lines

New! Debug with high-level source code
New! Redefine windows while debugging
New! View text files while debugging

- Debug device drivers, non-DOS and memory-resident programs

New! Customize Periscope via user exits
New! Display 8087/80287 status
New! Use Periscope with an EGA

Periscope requires: IBM PC, XT, AT, or close compatible; DOS 2.0 & later; 128K RAM; 1 Disk Drive; 80-column Monitor.

Periscope II, break-out switch, manual, reference card and software . . . $145!

Periscope I also includes the write-protected RAM board to protect crucial debugger code. It's just $295!

The US Navy gets Periscopes from us . . . shouldn't you? Order today!

Order/Information Call Toll-Free:

800-722-7006

30-Day Money-Back Guarantee

Data Base Decisions • 404/256-3860
14 Bonnie Lane • Atlanta, GA 30328

Circle no. **130** on reader service card.

NEW!

ZBasic:

Finally, the simplicity of BASIC, yet faster than Turbo.

- Works the same on all micros
- Uses same commands—regardless of computer make
- Structured Basic— (spaghetti optional)
- Device-independent graphics (same graphic commands on all computers)
- 6 – 54 digits of precision (selectable by user)
- Built-in interactive editor and compiler—to compile and execute, just type "run."
- Choice of alphanumerics labels or line numbers
- Chaining with shared variables

Not Copy Protected—No Run Time Fees or Royalties
One low price—No Hidden Costs. Only $89.95 complete!

•=YES	ZBasic Interpiler 3.0	TURBO PASCAL 3.0	MBASIC compiler	True BASIC	BASICA ~~Interpreter~~ (IBM PC)
IBM and compatibles	•	•	•	•	•
Apple //e, //c (6502)	•	Not Avail.	Not Avail.	Not Avail.	Not Avail.
Macintosh	4th Qtr.	Not Avail.	Not Avail.	Not Avail.	Not Avail.
CP/M-80 2.2, 3.0	•	•	•	Not Avail.	Not Avail.
TRS-80 Mod I, III, 4, 4p	•	Not Avail.	•	Not Avail.	Not Avail.
Direct commands	•	Not Avail.	Not Avail.	•	•
Maximum scientific digits of accuracy. (COS, SIN, ATN, LOG, EXP etc.)	6 to 54 selectable by the user	11 Binary BCD Not Avail.	16	16	6
Device Ind. Graphics (same commands all graphic modes and computers)	•	Not Avail.	Not Avail.	Not Avail.	Not Avail.
SAME File commands all computers?	•	Not Avail.	Not Avail.	Not Avail.	Not Avail.
STRUCTURED: Labels, Functions, LONG IF etc.	•	•	Not Avail.	•	Not Avail.
Same editor commands all versions/computers	•	•	Not Avail.	Not Avail.	Not Avail.
Sieve benchmark (Byte January 1983, 10 iter's)	13.7 sec.	14.1 sec.	14.9 sec.	261 sec.	2190 sec.
Shell-Metzner SORT (Sybex-BASIC for Scientist's and Eng. 2,000 5 char. strings)	19 sec.	28 sec.	71 sec.	194 sec.	2700 sec.
Executable Machine Lang. & approx. File size	12k	12k	32k	Not Avail.	Not Avail.
PRICE with BCD BCD=No rounding errors)	89.95	109.95	450.00	Not Avail.	Not Avail.
PRICE without BCD	89.95 BCD FREE	69.95	395.95	149.95	Comes with computer

Zbasic is a trademark of ZEDCOR Inc. © 1985. IBM is a registered trademark of International Business Machines Corp. Apple //e, //c are trademarks of Apple Comp. Inc. Macintosh is a trademark licensed to Apple Comp. Inc. PM-80 is a registered trademark of Digital Research Inc. TRS-80 is a registered trademark of the Tandy Corp. TURBOPASCAL is a registered trademark of Borland Int. MBASIC is a registered trademark of Microsoft Corp. True BASIC is a registered trademark of Addison-Wesley Publishing Co. BASICA is a registered trademark of International Business Machines Corp. All benchmarks and accuracies apply to standard IBM PC with 8088. Other computers and/or CPU benchmark speeds, accuracies and object code file sizes will vary depending on the computer clock speed, operating systems and other factors beyond our control.

ZBasic™

The lightening-fast BASIC from Zedcor

To order use this coupon or call (orders only): 1-800-482-4567
INQUIRIES: (602) 795-3996

(Name)
(Address) (Apt.)
(City) (State) (Zip)
(Day Phone)

CREDIT CARD—MASTERCARD/VISA/AMEX/C.O.D.

CREDIT CARD EXPIRATION DATE: ______

Mail to:
ZEDCOR INC
4500 E. Speedway, Ste. 93
Tucson, AZ 85712

$89.95 for each version

- ☐ IBM PC & Compatibles
- ☐ Apple //c, //e
- ☐ CPM-80
- ☐ KAYPRO Graphics
- ☐ TRS-80 (Model 1,3)
- ☐ TRS-80 (Model 4)
- ☐ Macintosh (4th QTR '85)
- ☐ DEVELOPERS—BUY ALL 6—$399.75. ($449.75 VALUE) SAVE $140.00

U.S. shipping $5.00 per order ($10.00 C.O.D.) $ ______

TOTAL $ ______

Arizona Residents add 5% Sales Tax. C.O.D. add additional $5.00 per order. Orders outside U.S. and Canada add $20.00 per shipment for postage (U.S. Currency only).

Circle no. **160** on reader service card.

Ram Disk Listing (Listing continued, text begins on page 84)

```
2682:                996 * REST OF BUFF MORE THEN NEEDED
2682:         0200   997 MIDBUFF    DS    512
 234B ABORTMSG        23F9 BE0           2409 BE1            FF3A BELL
 2235 BITMAP            D2 BLOCK1          D3 BLOCK2       ?   D4 BLOCK3
   D5 BLOCK4            D6 BLOCK5      ?   D7 BLOCK6             46 BLOCKNUM
 223E BM2               44 BUFFADR       23F1 BUFFERROR      2657 BUFFER
 2118 C1              211C C2            2124 C3             2126 C4
 2201 CB1             2215 CB2           2218 CB3            2277 CD0
 2286 CD1             2295 CD2           22AA CD3            22BA CD4
 21E5 CE              2250 CHECKDEV      222B CL1            2228 CLEARBUFF
 264A CLEARDIR        21E2 CLEAREND        42 COMMAND       ?210C COMPUTE
 2169 CONT            FDED COUT          FD8E CROUT            8D CR
 22D9 DA1             22E9 DA2           22D1 DEVABORT       BF1A DEVADR51
 BF31 DEVCNT          BF32 DEVLST          06 DIRBLOCKS      22EF DIRFOUND
 2564 DISPVOL         2566 DV0           2576 DV1            2578 DV2
 258B DV3             218F ENDSSD          00 EOL            217E EX1
 218E EX2             217A EXIT          BEF5 GETBUFF        2541 GETKEY
 2544 GK1             FC58 HOME          C000 KEY            2190 MAIN
 2682 MIDBUFF         264B MLIWRITE      BF00 MLI           ?2651 MLIREAD
 21B3 MN1             259B MSG           24B3 NOBUFF         D0A2 NODEV
 22CB NODIR           21B6 R1              48 RAMBANK       ?2655 READBLOCK
?2653 READBUFF       ?2156 READBYTE        01 READ           21B4 RELOCATE
 2106 S1                08 SAVEBYTES     21C8 SD1            213C SETBLOCK
 21C1 SETDEV          21D7 SETVECTOR       02 SSDCODE        C0D1 SSDHI
 214C SSDIO           C0D0 SSDLOW        2100 SSDSTART       C010 STROBE
   4A TEMP          ?   00 TEMPZERO      2554 TI1            2563 TI2
 254F TITLE         ?   43 UNITNUMB      2631 VOLNAME        265C VOLUME
 BE00 WARMDOS         240F WHEREMSG      2397 WHERETO       ?  02 WRITE
 264F WRITEBLOCK      2161 WRITEBYTE     21EF WRITEDIR       239C WT0
 23AC WT1             23CB WT2           23DA WT3            23E8 WT4
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 14-JUN-83 09:55
** TOTAL LINES ASSEMBLED    997
** FREE SPACE PAGE COUNT    87
```

End Listing

No source code for your REL files?

REL/MAC

converts a REL file in the Microsoft™ M80 format to an 8080 or ZILOG™ Z80 source code MAC file with insertion of all public and external symbols.

- **REL/MAC** makes MAC source files
- **REL/MOD** lists library modules
- **REL/VUE** displays the bit stream
- **REL/PAK** includes all of the above
- 8080 REL/MAC **demo disk** $10.00

REL/PAK for 8080 only $99.95
REL/PAK for Z80 & 8080 .... $134.95
on 8"SSSD disk for CP/M™ 2.2

Send check, VISA, MC or C.O.D. to

MICROSMITH COMPUTER TECHNOLOGY
P.O. BOX 1473, ELKHART, IN 46515
1-800-622-4070
(Illinois only 1-800-942-7317)

Circle **no. 103** on reader service card.

ICs PROMPT DELIVERY!!!
SAME DAY SHIPPING (USUALLY)

OUTSIDE OKLAHOMA: NO SALES TAX

640 Kbyte MOTHERBOARD KITS: IBM PC/XT Zenith 150 & 100, Compaq Portable & Plus: $65.00

V20	CPU μPD70108D-8		$16.00
8087-2	Math Coprocessors		140.00
DYNAMIC RAM			
256K	64Kx4	150 ns	$8.45
256K	256Kx1	120 ns	3.25
256K	256Kx1	150 ns	2.47
128K	128Kx1	120 ns	5.20
64K	16Kx4	150 ns	2.75
64K	64Kx1	150 ns	1.00
EPROM			
27C256	32Kx8	250 ns	$9.99
27256	32Kx8	250 ns	4.75
27C64	8Kx8	150 ns	3.99
2764	8Kx8	250 ns	2.25
STATIC RAM			
6264LP-15	8Kx8	150 ns	$3.25

QUANTITY ONE PRICES SHOWN
NEW! 100 ns 256Kx1 D-RAM @ 5.15

OPEN 6½ DAYS: WE CAN SHIP VIA FED-EX ON SAT.

ORDERS BY Th: Std Air Fr: P-One NO EXTRA COST ON F-EX SAT DELIVERY

MasterCard/VISA or UPS CASH COD
Factory New, Prime Parts μP∞
MICROPROCESSORS UNLIMITED, INC.
24,000 S. Peoria Ave., BEGGS, OK. 74421 **(918) 267-4961**

Prices shown above are for Nov. 18, 1985

Please call for current prices. Prices subject to change. Please expect higher or lower prices on some parts due to supply & demand and our changing costs. Shipping & insurance extra. Cash discount prices shown. Orders received by 6 PM CST can usually be delivered to you **by the next morning, via Federal Express Standard Air @ $6.00, or Priority One @ $13.00!**

Circle **no. 105** on reader service card.

HISOFT

HIGH QUALITY SOFTWARE **CP/M**

HiSoft has been selling Z80 CP/M software in Britain and Europe for over 4 years. Now we'd like to introduce **you** to our range of programming languages:

HiSoft Devpac: Z80 assembler/editor/debugger
HiSoft C: Kernighan/Ritchie implementation
HiSoft Pascal: fast, **standard** compiler

All at $69 inclusive each.

These programs are also available for other Z80 machines including Timex 2068.

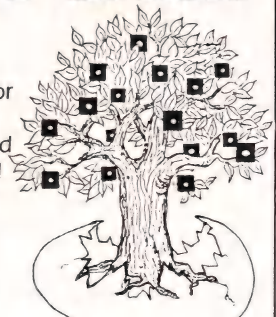
Call or write for full technical details and press commentaries, or order from:

HISOFT
180 High St. North
Dunstable LU6 1AT
ENGLAND
01144 (582) 696421

Circle **no. 134** on reader service card.

Tree Shell

A Graphic Visual Shell for Unix/Xenix End-Users and Experts Alike!



Dealer inquiries welcomed.

COGITATE

"A Higher Form of Software"
24000 Telegraph Road
Southfield, MI 48034
(313) 352-2345
TELEX: 386581 COGITATE USA

Circle **no. 81** on reader service card.

ASSEMBLERS

We will support you on over 20 UNIX and Xenix based machines. Targets:

Fairchild	F8/3870
Hitachi	6301, 6305, HD64180
Intel	8041, 8048, 8051, 8080 8086 family
Motorola	6800, 6801, 6805, 6809 68HC11, 68000 family
NSC	800
RCA	1802
Rockwell	6502/65c02
Texas Inst.	TMS7000
Zilog	Z8, Z80

UNIWARE™

SOFTWARE DEVELOPMENT SYSTEMS, INC.
3110 Woodcreek Drive
Downers Grove, IL 60515 U.S.A. (312) 971-8170
England: Unit C, Ltd. (0903) 205233

UNIX is a Trademark of AT&T Bell Labs
Xenix is a Trademark of Microsoft

Circle **no. 115** on reader service card.

UNIVERSAL CROSS-REFERENCER

- **IT WORKS WITH ALL LANGUAGES**
BASIC, C, PASCAL, FORTRAN, COBOL, ASSEMBLER, dBASE . . . you name it.
- **IT HANDLES** standard languages, extended languages & exotic languages.
- **IT CONFIGURES** to your compiler, interpreter, or assembler — ALL BRANDS — ALL VERSIONS.

INTRODUCTORY PRICE **$39.95**

$3.00 S/H - MC/Visa/Check - 6% Texas Sales Tax
For the IBM PC, XT & compatibles. DOS 2.0+

DALSOFT SYSTEMS
3565 High Vista - Dallas, TX 75234
(214) 247-7695

Circle **no. 86** on reader service card.

THE HAMMER
Software Tools in C

"I have already saved weeks of coding . . . thank you for providing such a useful tool . . . " G.T.

Let **The HAMMER Library** of over 150 routines save you valuable development time and effort:

LIBRARY FUNCTIONS
- **Multi-level 123-like MENUS**
- **DATA ENTRY**
  - MULTI-FIELD mode for Full-Screen data entry
  - Single-Field mode for individual fields
  - Data Verification
  - Full Editing within each field
  - Strings, dates and fixed decimal numbers
  - "Option" fields force user to pick from a given set
- **SCREEN MANAGEMENT**
  - cursor positioning
  - display boxes & tables
  - full attribute control
  - scrolling and clearing
- Date/time/string conversions
- BIOS access/pattern matching/**and more**

SUPPORTS: Microsoft C 3.00 • CI-C86 • M W C86 • DeSmet C • Lattice

INCLUDES source code and manual — $195 plus shipping VISA/MC accepted

O.E.S. SYSTEMS
1906 Brushcliff Rd. • Pittsburgh, PA 15221
412/243-7365

Circle **no. 137** on reader service card.

FORTH
and Assembler for
65802/816*

16 bit Processing for Upgraded 6502 Systems

Forth, Assembler, and Debugger for 65 Series Processors On a VAX or PC*

Starlight Forth Systems
15247 N. 35th St
Phoenix, AZ 85032
(602) 992-5181

*65802 on Standard 6502 systems

Circle **no. 142** on reader service card.

C Users' Group

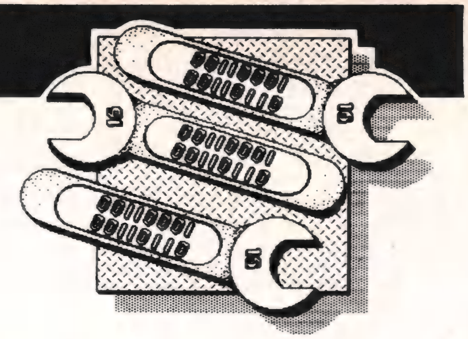
Over 60 volumes of public domain software including:

- compilers
- editors
- text formatters
- communications packages
- many UNIX-like tools

Write or call for more details

The C Users' Group
Post Office Box 97
McPherson, KS 67460
(316) 241-1065

Circle **no. 63** on reader service card.



by Ray Duncan

The programs published in this month's column are available for downloading from the Laboratory Microsystems RBBS at (213) 306-3530 (300 baud or 1200 bps).

Microsoft Assembler

I never cease to be astonished at the flurry of letters that results every time we publish something about the Microsoft Macro Assembler.

J. C. Hoisington of Morgan Computing (publisher of Trace86 and Professional Basic) writes: "We switched from IBM to Microsoft at Version 2.0. That means we didn't get SALUT but we did get to upgrade to Version 3.0 early this year for $75.00. Version 3.0 finally corrects many of the errors (and ridiculous restrictions) of earlier versions. The most useful to us is that they finally allow

```
lods byte ptr es:[si+0]
```

"which assembles as 26AC. I still found many bugs in Version 2.0, but 3.0 seems to have very few glitches."

David L. Rabbers, of Seattle, Washington, writes: "Regarding the errors [published in previous columns] two of them are yours, not the assembler's. Specifically, any constant ending in b or B is always treated as binary, and any constant ending in d or D is always treated as decimal. This is true regardless of the current radix setting. So, if you set the radix to 16, you must see to it that hex constants ending in b, B, d, or D are followed by an h or H. A little inconvenient, but how else could you specify binary or decimal constants in the midst of radix 16?

"One feature of Microsoft MASM Version 3.0 is a new linker that supposedly supports simple overlays. Unfortunately, there is no hint that I can find on how to use this feature. If you know how, or discover how, *please* publish it." Good point. My manuals don't breath a word about the construction of overlays. Anyone out there have some inside information on this subject?

Robert A. Blair, of Walnut, California, made the same point about radix overrides and went on to say: "This assembler should be used in Computer Science courses in college as an example of the wrong way to design a compiler; it is the worst I have ever seen. If you thought that Version 1 had a zillion bugs, wait until you use Version 2 for a while.

"I have to define my data before I use it (I thought I only had to do that in Pascal) or explicitly tell the compiler where it is to avoid phase errors. A multi-pass compiler should be able to figure out where the data is located without my having to tell it. The accompanying assembly listing shows a few errors that I know about. The 80286 PUSH immediate instruction generates the wrong length for some values. It will generate an instruction that requires two bytes of immediate data and only inserts one byte, or generates an instruction that requires one byte of data and inserts two bytes. The OFFSET operator causes link errors occasionally. The MASK used with HIGH does not work . . . These are just the tip of the iceberg." Robert's contribution accompanies the column as the listing (page 106).

Converting to DOS 3

Dan Daetwyler is a regular correspondent to *Doctor Dobb's* and always has something interesting to say. This month he shared some of his experiences with the latest version of DOS: "When DOS 2.0 introduced the concept of a file handle, I looked it over fairly carefully, and then decided we were being shown 'the way to go.' I went. It was a nice concept, and effectively the number of file handles that could be used at one time was unlimited. They made it so you could not use more than 99 handles at once—a good, typical effort by programming to cope with the concept of infinity on a finite machine. This was all well and good, since no one was going to have more than 99 files going on a PC.

"Then along came the new Version 3.x DOS. Gee, it just *had* to be better (and bigger), and all that good stuff. I read the new features section, and there *were* a couple of goodies. I devoured them carefully, and put them to immediate use. I bought Version 3.0, even though each time I swear I'll wait for the x.1 release—I get so tired of fighting MS bugs. Still, I bought it, and the Technical Reference, and really thought I read the documentation thoroughly. Along came Version 3.1, for which I breathed a large sigh of relief, since I could now stop fighting DOS bugs and worry about my own. And then I hit the thing with one of my moderately large complex systems. And bounced like a rubber ball.

"Buried in the details of the FILE statement in the CONFIG dissertation is a neat little statement that they've arbitrarily reduced the number of active handles to 20! The constraint is for a process, which is interestingly undefined, but you can assume it's a DOS job or task. The horrible part is, they meant it!

"So 20 isn't too bad a constraint. There won't be too many people who want to exceed that limit. Well, on the same page there's a note that five of the handles are grabbed by DOS.

That's very true. So now you're down to 15. Still a reasonable number? Not in my estimation. Let's follow through on a reasonably complex data base problem. We've got several—let's say five—master files, each of which is indexed. The three primary files have double indices, while the less important two have only a single index. Five plus three times two plus two gives us thirteen. Still safe? Not really. As you're aware, the only way you can force a buffer purge in DOS is to pseudo-close a file. Doing this with a duplicate handle permits you to avoid the killing performance of a re-open. Let's be stingy. We'll only use one handle for this purpose. We've still got one left. However, I've yet to write a system that doesn't have at least three and sometimes half a dozen supporting files for archives, output, etc. The net result is that you end up short about five handles, even on a relatively small system."

Dan's idea of a "relatively small system" would give many of us nightmares. Note how Dan offhandedly supplies the reason for the existence of that mysterious DOS function 45H called DUP, whose "explanation" in the DOS Technical Manual is "Purpose: returns a new file handle for an open file that refers to the same file at the same position." I had always been perplexed by this function, and the Unix manuals weren't any more helpful than the DOS manual. By the way, the 5 handles preassigned by DOS to the Standard Devices aren't really lost. You can always close them, and then reuse them for some other file or device.

Dan writes on: "My current effort involves a minimum of 33 handles, and even that makes me uncomfortable. Goodbye to DOS Version 3.x. Except I can't! I can't really lock a customer into the old DOS. Especially with multi-tasking 'right around the corner.'

"If you think this tale has a happy ending, you're wrong. Let me tell you some of the things I've tried. You can set up a resident task, split the data base, and let the resident task own part of it. The catch is that if you try to communicate with the resident task, you're running under the main task control structure, and DOS will tell you that the access isn't authorized. I haven't figured out how DOS knows, as yet, so I can't figure out how to switch task structures. The traditional use of a user interrupt vector doesn't work. DOS senses this and tells you again that the access is not authorized. Setting the file up with the proper attributes for a shared resource, and loading SHARE, doesn't help. I guess I could fight through a disassembly of DOS and find out what they're doing, but I don't want to spend the rest of the summer on this.

"I started to drop back to my own FCBs, and drop the handle concept. There's no limit to the number of files I can open, then, but then I'm back to the problems of the pre-handle DOS, and I can't see any way to purge the buffers. I can't really see exposing my customer to loss of data base when the power drops unexpectedly. I can't actually give him total integrity, but using the pseudo-close purge gets the risk factor down to the noise level. I've three of the complex database

systems out now, and in use on about fifty machines. We're semi-rural, and the power fluctuation during thunderstorms has to be seen to be believed, yet I've never (knock wood) lost a database from that source, since I started using the pseudo-close.

"So right now, I'm stuck. I can't move two of my systems to DOS 3.x, and the new one isn't even in the ball park . . . So this tale ends with questions. Do you know anything of help in this area?"

Actually, even the DOS 2.1 manual (pages 4-10) said that the maximum number of handles that a process could have concurrently open was 20, but that the maximum for the system was 99. This apparently includes background tasks such as the print spooler. Perhaps they just didn't enforce the process maximum. The manual for DOS 3.1 specifies the maximum for the system can be in the range 8-255, but again the maximum per process is 20. For those of you who are still using FCBs for whatever reason, be warned that DOS 3.1 puts some new and fairly strict restrictions on the use of FCBs when the file sharing module is also loaded. The default maximum number of FCBs that can be simultaneously open is 4. If you try to open more than your maximum, DOS unilaterally closes the least recently used FCB. You can increase the maximum allowed number of simultaneously open FCBs by a command FCBS=nn in the CONFIG.SYS file, just as FILES=nn works for handles. This is an omen, I believe, that FCBs are as doomed as the dinosaurs.

Microsoft and IBM have issued particular warnings about use of FCBs in the networking environment. If you use an FCB to access a file and fail to close it (a very common thing to do when you are using the file for read only), the system can bog down due to excessive connections accumulating across the network. Also, they remark that moving FCBs around in memory after they are opened can cause problems.

Dan proceeds: "Tale two is slightly more amusing. When I got my AT, and then started hearing all the terror stories about the hard disk, I consid-

ered myself very fortunate when I didn't encounter any . . . for a while. Then I had to load a 5 Mbyte database, and my trouble started. After several days of frustration, I nailed my friendly dealer to the wall, and IBM emergency shipped a replacement hard disk, which made life a lot better. In the interim, I did several interesting things. I put a trap in the program so that any time the I/O failure occurred, I'd get control. I was running under DOS 3.0, of course. The first thing I discovered was that when the disk glitched, the error code that DOS gave me back had no relation to disk I/O at all. In fact, DOS kept insisting that I was 'out of memory.' On a half megabyte machine with one small application?—not bloody likely.

"The next thing I discovered was that if I simply tried the operation again, by looping back to the interrupt, 99% of the time the thing worked happily on the second try. Very interesting! Then I discovered by accident that I was running with VERIFY=OFF. So I changed to VERIFY=ON, and got a whole new set of behavior patterns. Different error codes, and the fail ratio went way down. Running in this mode, the problem seemed to be occurring when DOS encountered something it thought was a bad track, so I offloaded my 20 Mbyte disk, and reformatted. Oh boy. The plot thickens like glue. I could run FORMAT repetitively, and every time I ran it I would find a different set of 'bad tracks.' There didn't even seem to be a consistent pattern. I believe it did have a relationship to something, but as near as I could detect, it was the phases of the moon, or the position of Jupiter.

"I won't bore you with more of the experiments that cost me about ten days of time, but I will tell you the punch line. We all heard about the hard disk problems in all of the trade journals. IBM replaced hard disks left and right, and the disk vendor was the bad guy. Very interesting, since, when I got Version 3.1, all the problems disappeared. Now DOS gives back the correct error codes. Now FORMAT is at least consistent."

Dan's second tale is the latest of a raft of mutually exclusive explanations of the mysterious AT hard disk problems. We have heard that the original hard disk was no good because the vendor tried to ramp up production too fast (and a recent review of the CMI hard disk showed it to be much more shock-sensitive and error-prone than its competitors); that the FORMAT program had a bug that prevented it from actually checking the second half of the 20 Mbyte hard disk; that the disk controller had design or component problems; and so on. Throughout most of this time, the official IBM press flacks insisted that everything was wonderful and there was no AT disk problem at all. It certainly makes one apprehensive about the position we'll all be in if IBM manages to knock off all its competitors in the PC market.

DDJ

(Listing begins on next page)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 197**.

16-Bit Listing (Text begins on page 102)

```
                                                page    55,132
                                                title   Microsoft Assembler Bugs
                                                .286C

                                        ; Some IBM PC Macro Assembler 2.0 "features"
                                        ; contributed by Robert Blair

0000                                    foo     segment para public 'CODE'
                                                assume  cs:foo
0000                                    foos    proc    far
0000  E9 010B R                                 jmp     foobar
                                                                ; this bug causes link errors if
                                                                ; the offset is larger than 256.
                                                                ; the error is
                                                                ; 'fixup offset exceeds field width'
0003     02 [                                   dw      2 dup (offset foobar)
                010B R
                        ]


                                                                ; this will work ok
0007     02 [                                   dw      2 dup (foobar)
                010B R
                        ]


000B  0100 [                                    db      256 dup (0)
                00
                    ]


010B  CB                                foobar: ret

                                        ; it appears that if any calculations are required, the compiler
                                        ; truncates after the first two bytes.  The assembler reference
                                        ; manual p. 2-15 states this is done for the small assembler but it
                                        ; does it for both assemblers.

010C  80 51 00 00                               dd      60*60*24        ; incorrect
0110  80 51 01 00                               dd      86400           ; correct


                                        ; Now try HIGH and MASK operators together

                                        fooi    record  f1:1,f2:1,f3:3,f4:1,f5:1,f6:1,f7:8
= 0200                                  fc1     equ     0200h
= 4000                                  fc2     equ     4000h

                                                                ; immediate data should be
0114  25 0200                                   and     ax,mask f5                      ; 0200h
0117  25 4000                                   and     ax,mask f2                      ; 4000h
011A  25 0040                                   and     ax,high (mask f2)               ; 0040h
011D  25 0042                                   and     ax,mask f5+high(mask f2)        ; 0240h
0120  25 4200                                   and     ax,(high(mask f2))+(mask f5)    ; 0240h
0123  25 0200                                   and     ax,fc1                          ; 0200h
0126  25 4000                                   and     ax,fc2                          ; 4000h
0129  25 0040                                   and     ax,high fc2                     ; 0040h
012C  25 0042                                   and     ax,fc1+high fc2                 ; 0240h
012F  25 4200                                   and     ax,(high(fc2))+(fc1)            ; 0240h
0132  25 4200                                   and     ax,fc1 or high fc2              ; 0240h


                                        ; 80286 PUSH IMMEDIATE instruction bugs
                                        ; an op code of 6A requires one byte of immediate data
                                        ; an op code of 68 requires two bytes of immediate data

0135  6A 00                                     push    0               ; 6a00 correct
0137  6A 7F                                     push    127             ; 6a7f correct
0139  6A FF                                     push    -1              ; 6aff correct
013B  68 80                                     push    128             ; should be 680080
013D  68 0080                                   push    word ptr 128    ; generates correct code
0140  68 FF                                     push    255             ; should be 6800FF
0142  68 00FF                                   push    word ptr 255    ; generates correct code
```

```
        0145  6A FFFF                        push    0ffffh          ; should be 6aFF or 68FFFF
        0148  6A 34                          push    byte ptr 1234h  ; lost data with no error


                                    ; I have to define my data before use or tell the assembler
                                    ; where it is.  Multi-pass compilers should be smarter
                                    ; than this

        014A  2E: D7                         xlat    fred            ; causes (spurious) phase error
                                                                     ; message but generates right code

        014C  2E: D7                         xlat    cs:fred         ; works ok

                   FF [             fred     db      255 dup (0)
 E r r o r   ---          6:Phase error between passes
                          00
                             ]



                                    ; This bug is not very serious because the only thing wrong is
                                    ; the assembly listing.  The generated code is correct but the
                                    ; listing of the generated code is not very neat.

        024C  0D 3D [D 0A                    db      13,10,13,10,61 dup ('-'),13,10
                          2D
                             ]
                               0D 0A


        028F                        foos     endp
        028F                        foo      ends
                                             end     foos
```

End Listing

ANNOUNCING! DR. DOBB'S COMPLETE TOOLBOX OF C

Dr. Dobb's Journal, the most respected source of technical software information available, brings you this collection of powerful programming tools for C.

New, from M&T Publishing and Brady Communications . . .

Dr. Dobb's Toolbook for C

Item #005

A comprehensive library of valuable C code.

Many of Dr. Dobb's most popular articles on C from sold-out issues are updated and reprinted in this unique reference, along with new C programming tools. **THE TOOLBOX** contain's C-compilers, line editors, assemblers, text processing programs, and more! Dr. Dobb's Journal offers **The Toolbook** in a special hardbound edition for only $29.95. You'll find:

- J.E. Hendrix's famous *Small C Compiler v. 2* and *A New Library for Small C* (also available on disk)
- Never before published: Hendrix's new *Small-Mac: An Assembler for Small C* and *Small Tools: Programs for Text Processing* (both available on disk)
- Ron Cain's original *A Small-C Compiler for the 8080's*
- Plus many useful programming tools in C

Also from M&T Publishing and Brady Communication—

The Small-C Handbook

Item #006 or #006A

The Small-C Handbook by James Hendrix is a valuable resource of information about the Small-C Compiler. In addition to descriptions of the language and the compiler, **The Handbook** also contains the entire source listings of the compiler and its library of arithmetic and logical routines. A perfect companion to the Hendrix Small-C compiler offered by Dr. Dobb's on disk, **The Handbook** even tells how to use the compiler to generate new versions of itself. All this is in **The Handbook** for only $17.95, Item #006; only $22.95 for **The Handbook** with the **MS/PC DOS Handbook Addendum,** Item #006A.

While both **The Toolbook** and **The Handbook** provide documentation for the Small-C Compiler on disk, **The Handbook** provides more complete documentation and is available with an Addendum for MS/PC DOS versions.

DR. DOBB'S C TOOLS ON DISK

To complement The Toolbook, Dr. Dobbs' also offers the following programs on disk. Source code is included, and except where indicated, both CP/M and MS/PC-DOS versions are available.

Small C Compiler

Item #007

Jim Hendrix's **Small-C Compiler** is the most popular piece of software published in Dr. Dobb's 10-year history. Like a home study course in compiler design, the **Small-C Compiler** and **The Small-C Handbook** provide all you need to learn how compilers are constructed, as well as teaching the C language at its most fundamental level. The **Small-C Compiler** is available for $19.95 in both CP/M and MS/PC DOS versions.

Both **The Toolbook** and **The Small-C Handbook** provide documentation for the compiler; however, **The Handbook** provides more complete documentation for both versions, and the **MS/PC DOS Small-C Handbook Addendum** is recommended in addition to **The Handbook** for MS or PC DOS specific documentation.

Small-Mac: An Assembler for Small-C

Item #012A

Small-Mac is a macro assembler designed to stress simplicity, portability, adaptability, and educational value. The package features simplified macro facility, C-language expression operators, descriptive error messages, object file visibility, and an externally defined machine instruction table. Included programs are: macro assembler, linkage editor, load-and-go loader, library manager, CPU configuration utility, and dump relocatable files. This program is available with documentation in the Small Mac Manual for $29.95. For CP/M systems only.

Small Tools: Programs for Text Processing

Item #010A

This package consists of programs designed to perform specific functions on text files, including: editing, formatting, sorting, merging, listing, printing, searching, changing, transliterating, copying and concatenating, encrypting and decrypting, replacing spaces with tabs and tabs with spaces, counting characters, words, or lines, and selecting printer fonts. Source code only is included. This program is available with documentation in the Small Tools Manual for $29.95. Both CP/M and MS/PC DOS versions are available.

Special Holiday Packages—20% Off!

Now, for almost 20% off the individual product prices, you can order a complete set of Dr. Dobb's C programming tools for your CP/M or MS/PC DOS system!

CP/M C Package Only

$99.95 Item #005A Ordered individually, these C tools sell for over $120! Order the CP/M C Package now and you'll receive **Dr. Dobb's Toolbook, The Small-C Handbook, The Small-C Compiler** on disk, **The Small Tools TextProcessor** on disk, along with documentation in **The Small Tools Manual, The Small-Mac Assembler** on disk, and **The Small-mac Manual**—all for only $99.95!

MS/PC DOS C Package Only

$82.95 Item #005B Individually, these tools sell for over $100! Order now and you'll receive: **Dr. Dobb's Toolbook, The Small-C Handbook** with the **MS/PC DOS Addendum, The Small-C Compiler** on disk, **The Small Tools Text Processor** on disk, along with documentation in the **Small Tools Manual**—all for only $82.95!

		Check Format CP/M	Check Format MS/PC DOS		
Books	Dr. Dobb's Toolbook			$29.95	
	Small-C Handbook			$17.95	
	Small C Handbook with MS/PC DOS Addendum			$22.95	
Disks	Small-C Compiler (Toolbook or Handbook recommended for documentation)			$19.95	
	Small Tools Text Processor with Small Tools Manual			$29.95	
	Small Mac Assembler with Small Mac Manual (for CP/M only)			$29.95	
SPECIAL HOLIDAY PACKAGES	CP/M C Package			$99.95	
	MS/PC DOS C Package			$82.95	
				Subtotal	
				Tax ___ %	
				Shipping	
				TOTAL	

CA residents add applicable sales tax
Add $1.75 per item for shipping in U.S., $4.25 outside U.S.

For CP/M system disks only, please specify one of the following formats:
☐ Kaypro ☐ Apple ☐ Zenith Z-100 DS/DD ☐ Osborne ☐ 8″ SS/SD
Inquire about other formats.

PAYMENT MUST ACCOMPANY YOUR ORDER.

___ Check Enclosed
___ Charge my ☐ VISA ☐ M/C ☐ Amer. Exp.

Card # ______________ Exp. ______

Signature ______________

Name ______________

Address ______________
(Please use street address)

City ______________ State ______ Zip ______

Please return this coupon along with your payment to: Dr. Dobb's Journal, 2464 Embarcadero Way, Palo Alto, CA 94303 For credit card orders, call toll-free: 1-800-528-6050 Ext. 4001 Refer to item numbers.

Please allow 4-8 weeks for delivery

3110A

Introduction to Using Your MacSCSI Host Adapter

by John Bass

The MacSCSI host adapter was designed to work with most SASI- and SCSI-compatible controller cards. It is also a reasonable starting point for other custom Mac interfaces. Source software written in Aztec C is provided for custom configuring your system. If your controller is Xebec S1410 compatible you should be up and running with little trouble.

Changing the Parameters for the Drive and Controller

For controllers other than the Xebec S1410, you may need to modify the last byte of the format, read, and write commands to reflect the vendor-specific requirements. The Xebec S1410 uses this byte to encode the step rate for the drive—as shipped, the code is set up to use a Seagate ST506 drive with the half-step strap enabled. If you are using a Xebec S1410 with another drive, consult your controller manual for the correct drive code to use. Then change the cmd.sc_vendor assignment from 1 to the correct drive-type code.

Because the Xebec defaults to an ST506-drive configuration we didn't need to issue any additional commands to define the drive configuration to the controller. This is an area that is very vendor specific—read your controller documentation very carefully. You will want to read the descriptions for the format, read, and write commands closely. Also check the error-code definitions carefully because they vary greatly between some controllers.

You may need to issue a Set Media or Set Mode command that defines the drive parameters for number of heads, cylinders, write precomp, write reduce, and so on. Some drives, such as the Xebec Owl, have an embedded controller and don't need such a setup parameter. Some controllers encode the parameters at format time and write them on the disk for future use. Other controllers require a Mode Select or other command to set the drive parameters prior to the first format, read, or write command.

The long evolution from early SASI controllers to the current proposed SCSI specification has left a lot of nonconforming controllers around the marketplace, and many more nonconforming controllers are likely to be produced before the standard is finished and adopted. Until then, the software drivers for each host adapter will need to be customized per controller.

A copy of the current SCSI (ANSC X3T9.2) specification is available from the American National Standard publications office. For NCR-5380 information contact your local NCR Microelectronics representative or write to the NCR Microelectronics Division, 1635 Aeroplaza Dr., Colorado Springs, CO 80916; (800) 525-2252.

Mounting Drives Inside or Outside the Mac

Several 5-, 10-, and 20-megabyte 3½-inch drives are now on the market, and controllers in the same form factor are available as well. The easiest way to mount a 3½-inch drive, controller, and power supply internally is to fabricate a sheet metal and/or Plexiglass mount and screw it inside the plastic back of the Mac. To get AC for your power supply, install a power cable to the video board after the switch and AC filters. Be sure to leave a long enough service loop on both the power and SCSI cables to make getting the back off easy.

Other internal mounting configurations require more attention to sheet metal design and choice of controller, drive, and power supply. Convection cooling can be enhanced by mounting everything on the right side and using the sheet metal and Plexiglass to form a chimney extending from the bottom to the vents at the top of the case. EMI fields from the power supplies, flyback, and yoke coils present some problems in getting the disk to operate without errors—proper selection of shielding material and position is critical.

The easiest way to interface drives to the MacSCSI is to mount them externally in a separate box, so that physical mounting, heat, and EMI problems are less severe. Several manufacturers have standard line enclosures. Some of these enclosures are complete with power supply, cables, fans, and maybe a drive of your choice.

External drives require an external cable. The SCSI standard specifies an EMI shielded cable, plug, and header assembly. We use AMP part number 1-499977-0 for the panel-mounted connector and AMP part number 102793-4 for the ground plane (shield assembly) that mates with the connector. Use the ground plane only if you plan to use a SCSI standard shielded cable—it will interfere with a standard nonshielded IDC ribbon-cable connector.

Installing the MacSCSI Board

Getting the back off a Mac requires a Torx 5 screwdriver with a long shaft. There are five screws: two are located at the top in holes under the handle, two are on the back toward the bottom above the connectors, and one is located behind the battery cover. Carefully use a wooden ruler or the battery cover to ease the crack along

the front seam apart—do not use a screwdriver as a pry because the case plastic is very soft and will show pry marks.

Remove the power and floppy cables from the digital board, then gently slide it out of the sheet metal frame. Place the board on a table with the keyboard connector toward the back and the other connector near you. Lay the MacSCSI board near the ROMs with the SCSI header near you. Remove and transfer both of the Mac ROMs onto the MacSCSI board, then plug the MacSCSI board back into the ROM sockets.

Now clip the miniplug onto pin 8 of the TSM PAL located at D1 on the right edge of the board. Lay the clip flat against the board alongside the 15.667200 crystal and secure it with a piece of masking tape across the crystal and over the edge of the board. Route the wire between the switches and toroid coil and continue alongside the metal shield to the MacSCSI board. Replace the digital board in the frame, flexing slightly to pass the MacSCSI header past the edges of the frame. Be careful not to pull the board out of the ROM sockets. Cable the SCSI port as required, reconnect the power and floppy cables, and reassemble the case.

Bringing Up the System for the First Time

Once you have the system cabled up and have made any required software changes, try formatting the drive. Watch for the drive to be selected. On some drives you can see the positioner step from track to track, and on others you may be able to feel the drive stepping. Formatting takes a while—for 5- and 10-megabyte drives it may take several minutes, and 100-megabyte drives may need a half hour. The format routine initializes the Mac file system by writing a volume header and clearing all other blocks and then rereads all blocks to check for errors.

If you have problems with Aztec C you can add printfs to the C I/O routines and follow the progress. Once you have the format routine and I/O routines running, compile and link the driver with the I/O routines.

If you are connecting to something other than a disk you can simply include the I/O routines in your application and drive the SCSI bus directly.

Miscellaneous Comments

We have used MacSCSI under Finder 4.1 and Aztec C Shell 1.06D and F with a nonpartitioned 5-megabyte drive. Aztec C Shell and environment seem to run well even with a very large number of files. Finder has some limits you should keep in mind. On large volumes it gets pretty slow after 150 files. Finder also crashes sometimes when too many files get created. Deleting some files or the desktop under Aztec C sometimes makes it happy; at other times it seems that you need to back up the disk under Aztec C Shell and reformat the volume. We are working with

Apple to find the solutions to these problems.

We have used the driver with and without a cache with some interesting results. For small cache sizes the overhead of maintaining the cache slows the overall operation down. For the Aztec C environment the minimum size is roughly the sum of all the programs, libraries, and include files in use—it is easier to start up the RAM disk and copy everything to it. For large cache sizes it is a pain to run large programs or the switcher because you need to restart the driver with a smaller cache size. The overall conclusion for our working environment is that, although the cache can create RAM-disk speeds, it sometimes creates floppy speeds. The MacSCSI hard disk is nearly as fast as the RAM disk anyway, and maintaining the cache just becomes a headache when switching environments.

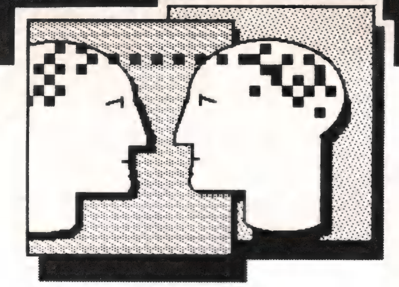
Another minor annoyance is that you can't remount the MacSCSI volume if you eject it from an open box—never do so. We are working on adding the code to trap the ejects and ignore them. The eject solution, multiple volumes via partitions, and other improvements will be available in our 2.0 software release within a few weeks.

We are collecting changes for other controller-and-drive combinations to be available on our update disk. If you would like to share your changes, send us a disk with your modifications and we will return it with the current updates. Mail your changes to Fastime, P.O. Box 12508, San Luis Obispo, CA 93406.

DDJ

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 198.**

CP/M EXCHANGE



by W. E. Wilson

Software running on a Z80-based system under CP/M-80 will frequently perform better than comparable software running on an IBM PC under MS DOS. For example, Microsoft BASIC-86 is actually slower on the IBM PC than BASIC-80 on a Z80-based system. Programs running under CP/M-80 also are frequently smaller than their counterparts on MS DOS. Translate a Z80 assembly-language program into 8086 assembly language and watch the code expand.

Sixteen-bit systems do have a number of advantages over their 8-bit predecessors. You can address more memory and can have better, even color, graphics. The PC can communicate easily with mainframe systems and will probably be more compatible with future developments than the older 8-bit systems. On the other hand, there are still many more software packages available for CP/M than there are for MS DOS, some quite good and very reasonably priced.

When comparing Z80-based systems to the IBM PC, there are some basic facts that you should keep in mind. First of all, the limited addressable memory space of 8-bit systems used to force programmers to write tight, rapidly executing code. With the advent of 16-bit systems and more addressable memory space, software developers were no longer compelled to keep code size small, and they shifted to less efficient programming techniques, including writing in system-independent, high-level languages. Second, most popular 16-bit programs were originally written for 8-bit systems. Developers have often moved these programs to the PC by translating the 8-bit code line by line into the equivalent 16-bit code. The result is that much of the potential efficiency of the 16-bit systems has been lost. For example, many of the spreadsheets that run without any problems on a 64K 8-bit system require more than 128K on a 16-bit system. Third, the memory overhead for the MS DOS operating system is so much greater than the overhead for CP/M-80 that a 128K PC is approximately equivalent to a 64K CP/M-80 system. Finally, the 5¼-inch floppy disk drives on a PC are as slow as molasses in January when compared to the typical 8-inch drives on the older CP/M-80 systems.

W. E. Wilson, Washington State University, Nuclear Radiation Center, Pullman, WA 99164

Mini System Software for a Micro

Back in the days before the invention of the processor on a chip and the development of modern micro systems, memory was very expensive. Minicomputer companies such as Digital Equipment Company provided overlaying linkers for use with their computers. In the early '70's the institution where I worked had a DEC PDP-15 with 64K of memory. DEC provided a software package called Chain and Execute that allowed you to run a very large program on the PDP-15 by segmenting it into a series of overlays. I used Chain and Execute to run some huge programs written in FORTRAN for the IBM 360 on the little DEC system. Not very much is written on the subject of overlays[1], but this technique is used extensively in mainframe systems and in multiuser operating systems.

The Linking Process

The creation of an executable program from a number of source files involves several steps. First, the source files are individually read by the compiler and converted into one or more object modules. Some compilers produce relocatable object code directly. Small-C and its derivatives generate assembly-language output that must be assembled. The following discussion applies to compilers of the latter type only.

It is not practical to have the compiler generate all the code needed for each module. Instead, code is generated only for operations specific to the particular module. General-purpose functions and subroutines from the run-time library are merely referenced by the compiler and only later linked into the program. These external references are "place holders" into which absolute addresses are eventually placed. For functions and variables defined within the module, the compiler generates references relative to the beginning of the module. This means that if the module is eventually loaded at address L, then all references will be correct if L is added to them.

An assembler then reads each individual module created by the compiler and generates for each one a relocatable object module. This module is called relocatable because it can be relocated anywhere in memory.

The final stage is linking. The linker searches the run-time library for any referenced subroutines and functions. It loads these subroutines and functions along with the relocatable object modules into their final positions in contiguous blocks of memory. Finally, it fills in the absolute addresses for externals and adjusts the relative addresses for internals.

A standard linker builds programs in memory. Thus, the maximum program size is equal to the total memo-

ry space minus the space used by the operating system and the linker. An overlaying linking loader links the program on disk rather than in memory and thus can link a program that will fill the entire available memory space or more. There is, however a penalty paid in speed. An overlaying linking loader must read every file twice—once to locate the globals and determine the size of the module and once to do the actual linking.

ZLKO

ZLKO from ZEE MICROWARE is an overlaying linking loader for Z80-based systems. The most important feature of ZLKO comes into play when a program is too big to fit into available memory space. The standard solution to this problem is to use the call-next-program technique. This technique involves dividing the program into a selected number of segments, each small enough to fit individually into memory. Each sequential segment except the last must load and execute the next segment. This method is only supported by a few compilers and has a number of disadvantages:

- It is not possible to return to a calling segment
- All segment intercommunication must be via disk files
- Selective segment execution is not possible

All of these weaknesses can be overcome by using the overlaying capabilities of ZLKO.

The user segments the program into a small root (main) module and one or more overlays, each of which is composed of a number of subroutines and functions. The subroutines are grouped into overlays according to criteria selected by the user. ZLKO then constructs a multilevel "tree" of overlays in which the overlays at any given level can call overlays at the next lower level. The execution of the segmented program begins when the root module is executed. The root will then activate other overlays by calling the subroutine "ovrlay" that has been linked with the root segment. The root segment will stay in memory

0100–0102	JP ENTRY
0103–0107	Identification Code
0108–0109	End + 1 of Last Overlay or first free memory location above overlay system
010A–010B	End + 1 of Root Segment or the start of first overlay
010C–0114	Eight Character Root File Name

Byte 0–1	Overlay Transfer Address (Load Address)
Byte 2–3	End + 1 of Segment
Byte 4–11	Eight Character Segment File Name

Figure 1
Root and Overlay Segment Headers

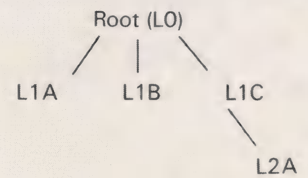


Figure 2
Structure of the test overlay system

```
ROOT(MAIN);OVINIT,OVRLAY,L0;
END;
SYMFIL(CTEST);
OVL(1,L1A);
INCL(L1A);
END;
OVL(1,L1B);
INCL(L1B);
END;
OVL(1,L1C);
INCL(L1C);
END;
OVL(2,L2A);
INCL(L2A);
END;
DONE;
```

Figure 3
ZLKO Command File Used to Create Overlay System

ZLKO Command Summary

CMND	= Take Command Input From Specified File
DCOM	= Display Common Block Information
DMAP	= Display Global and External Symbols
DONE	= Complete Current Segment and Quit
DOVL	= Display Overlay Structure
DSEG	= Display Information On Current Overlay Segment
DUND	= Display Undefined Symbols
END	= Complete Current Segment, forces a SRCH
HELP	= Display Command List
INCL	= Include The Listed REL Files in The Current Segment
OVL	= Build Overlay (.OVL) File
PBSE	= Set Program Code Base Location
QUIT	= Quit and Return to CP/M
RELDMP	= Dump Relocatable (.REL) File
ROOT	= Build ROOT (.COM) File
SRCH	= Search Specified (.REL) Library
STOP	= Quit Building Current Segment and Save The Link Table
SYMFIL	= Generate Symbol File For Symbolic Debugger
?SYM	= Suppress Symbols Starting with ?
TCSW	= Turn Off Module Counting

Figure 4
ZLKO Help Command Output

the entire time. The overlays will be loaded into memory and control passed to each as program execution proceeds. When an overlay returns control to the higher level that activated it, the higher level may then activate another overlay beneath itself or pass control down one level. Each overlay on the same level in the tree is loaded into memory starting at the same location.

Routines in an overlay can reference any symbol defined within that overlay and all global symbols located in any active higher-level overlay up to and including the root segment. Global symbols and routines in the root segment may be referenced from any segment. Frequently needed routines and functions should be included in the root segment. Routines in an overlay cannot, however, reference symbols or routines defined in an overlay on the same level or in a segment lower down the tree.

ZLKO Usage

ZLKO places a 21-byte header on the root-segment file and a 12-byte header on each overlay-segment file (see Figure 1, page 116). The headers include the primary CP/M file name of the segment as well as address information. In the case of an overlay segment, the overlay name and transfer address are used by the ovrlay subroutine to load the overlay. The execution process starts at the root segment and proceeds by descending down and climbing back up each branch of the overlay tree structure one branch at a time.

Using ZLKO with C offers one significant advantage over using it with a language like FORTRAN. FORTRAN employs the call-by-reference technique. Hence, arguments cannot be passed to a called overlay except by means of the COMMON block in the root segment. On the other hand, C uses the call-by-value technique. Thus, values can easily be passed to a function in an overlay. C passes the arguments by pushing them on the stack, and the function in the called overlay simply pops them off the stack.

ZLKO does not automatically insert code into the root segment to set the

stack pointer because the required call to main( ) in the first routine in the root segment will take care of setting the stack pointer. That is, this operation is handled by the C compiler, which expands the call to main to include setting the stack pointer.

Most Small-C compilers use the symbol _memptr to designate the first free memory location above the program code space. The Q/C compiler I used to test ZLKO uses the symbol $MEMRY for compatibility with the L80 linking loader. Under ZLKO, the value to be used for $MEMRY or _memptr is stored in the 8th and 9th bytes of the root-segment header. This value must be extracted from the header and stored in location $MEMRY or _memptr in the compiled C program before program execution commences.

This operation is readily accomplished by using the ovinit function shown in Listing One (page 119). Ovinit must be linked into the root segment and must be the first function called from within main( ). The ovrlay function supplied with ZLKO must be linked into the root segment following ovinit. Each time an overlay is to be loaded, the user program must make a call to the ovrlay function using a pointer to the overlay name as the last argument of the ovrlay function call. The overlay test (see next section) demonstrates the use of the ovrlay function. Ovrlay will load the specified overlay at the transfer address specified in the header and then execute the overlay.

ZLKO Test

I tested the ZLKO overlaying linking loader with a simple overlay system written in C. The test program was compiled with the Q/C compiler and assembled with the CWA assembler distributed by The Code Works. The structure of the test overlay system is shown in Figure 2 (page 116) and the command file used to create the overlay system in Figure 3 (page 116). The code for the root segment and each overlay is presented in Listing Two (page 119). Listing Three (page 125) shows the output that ZLKO sent to the console during the test. The linking process took 126 seconds and the resulting overlay system executes in 11 seconds.

Using ZLKO

ZLKO is relatively easy to use once you understand how to pass to ovrlay a pointer to the overlay to be loaded by the root segment. There are, however, a couple of things you should keep in mind. First of all, because the input information required by ZLKO is extensive, the potential for keyboard errors is quite high. This problem can be solved by using a command file instead of direct input. For example, entering the command:

```
ZLKO CMND(CTEST);
```

causes ZLKO to take its input from the command file CTEST.ZLK. Second, you may have trouble getting accustomed to the fact that all commands require a combination of semicolon and carriage return as a terminator. Finally, although the manual adequately describes the proper use of ZLKO, it will not make

you an expert at using overlays. All of the commands accepted by ZLKO are shown in Figure 4 (page 116) in the form of a printed copy of the output of the ZLKO Help command. ZLKO is quite verbose and lets you know what is going on at all times. Part of the verbosity is because of the fact that ZLKO displays each component of a command string as it is being parsed.

Conclusion

ZLKO is an easy-to-use, relatively efficient overlaying linking loader that is particularly useful to anyone using a Small-C compiler. ZLKO will allow the user to run very large programs under CP/M-80 and to construct fancy software in which a menu-selection operation determines which overlay (program) is executed next. There are at least two other overlaying linkers available for use with CP/M-80, but they are much more expensive than ZLKO and only provide a few additional capabilities. ZLKO only supports a multiple-file overlay file system, whereas some others also support a single-file overlay file system that uses the random-access capabilities of CP/M-80. A single-file overlay system will execute slightly faster but is more difficult to construct and debug.

Do not discard your older 8-bit system if you can perform the tasks you need to accomplish by simply purchasing an overlaying linking loader such as ZLKO. ZLKO is available from Elliam Associates, 2400 Bessemer St., Woodland Hills, CA 91367.

Notes

[1] A good discussion of overlays is provided by S. H. Kaisler, *The Design of Operating Systems for Small Computer Systems*, John Wiley, 1983.

DDJ

(Listings begin at right)

Reader Ballot
Vote for your favorite feature/article.
Circle Reader Service **No. 199.**

CP/M Exchange (Text begins on page 114)

Listing One

```
----------------------------------------------------------------

/* Get addess of first free memory location from ROOT header
and put it in location $MEMRY. This version is for use with the
Q/C compiler.  Change $MEMRY to _memptr for use with small-c */

ovinit()
      {
# asm
        EXT     $MEMRY
        LD      HL,(0108H)      ;Get 1st free mmry addr
        LD      ($MEMRY),HL     ;Put in $MEMRY
# endasm
     }
----------------------------------------------------------------
```

End Listing One

Listing Two

```
------------------------------------------------------------------------

/*  Root program L0.C for testing ZLKO with Q/C */

#include <qstdio.h>

main()
    {
char *ovla, *ovlb, *ovlc;
ovla = "L1A     ";
ovlb = "L1B     ";
ovlc = "L1C     ";
/* initialize Root Segment For Overlays */
      ovinit();
        printf("ZLKO Overlaying Linker Test with Q/C\n\n");
        printf("Starting at level zero\n");
        printf("Call Overlay L1A\n");
```

(Continued on next page)

Listing Two

```
 /* Load Overlay L1A */
          ovrlay(&ovla);         /* Pass pointer to ovlname */
            printf("Returned to Level zero from Level one pgm L1A\n");
            printf("Next Call Overlay L1B\n");
 /* Load Overlay L1B */
         ovrlay(&ovlb);          /* Pass pointer to ovlname */
           printf("Returned to Level zero from Level one pgm L1B\n");
           printf("Next Call Overlay L1C\n");
/* Load Overlay L1C */
        ovrlay(&ovlc);           /* Pass pointer to ovlname */
          printf("Returned to level zero, Overlay Test Done\n");
        }
----------------------------------------------------------------------------

/* overlay L1A */
levella()
        {
        printf("At Level one, Overlay L1A\n");
        }

----------------------------------------------------------------------------

/* overlay L1B */
levellb()
        {
        printf("At Level one, Overlay L1B\n");
        }
----------------------------------------------------------------------------

/* overlay L1C */
levellc()
        {
```

(Continued on page 123)

 (Listing continued, text begins on page 114)

Listing Two

```
char *ovld;
int valuel, value2, sum;
ovld = "L2A      ";
        printf("At Level one, Overlay L1C\n");
        printf("Call Level two, Overlay L2A 5 times\n");
        /* Load Overlay 2A */
          valuel = 0, value2 =9;
          while (++valuel < 6)
        { ++value2;
  printf("At Level one Calling L2A, valuel = %d, value2 =
%d\n",valuel,value2);
        /* The pointer to the overlay name must be the last parameter */
        /* The ZLKO "ovrlay" routine uses the value in the HL register- */
        /* -to find the overlay name but does not alter the stack */
        /* The called routine must have a dummy last parameter */
          ovrlay(valuel,value2,&ovld); }
          /* Pass values and pointer to ovlname */
        }
int sum;
retval(sum)
        {
        printf("Back at Level one, Returned sum valuel + value2 =%d\n\n",sum);
        }

------------------------------------------------------------------------------

/* overlay L2A */
level2a(argl,arg2,arg3)
        /* argl is valuel, arg2 is value2, and arg3 is ovlname pointer */
        /* dummy arg3 is required to match ovrlay function call in L1C */
int argl, arg2, arg3;
        {
        printf("At Level two, Overlay L2A\n");
        printf("Received parameter valuel =%d, value2 =%d\n",argl,arg2);
/* Pass sum argl + arg2 back to level one*/
        retval(argl + arg2);
        }
```

End Listing Two

(Listing Three begins on page 125)

CONTINUING THE TRADITION

DR. DOBB'S JOURNAL ANNOUNCES THE RELEASE OF BOUND VOLUME 8

Every 1983 issue together in one source

BOUND VOLUME 8 (Item #020)
DDJ turns pro. Some of the most powerful, professional programmer's tools ever published in a magazine are in this volume. Jim Hendrix's Small C compiler. Ed Ream's RED screen editor. A microcomputer subset of the Defense Department's official programming language, Ada. C and Forth and 68000 software. Because the magazine increased in size in 1983, this volume is bigger and better that ever.

Vol. 1 1976 (Item #013)
The material brought together in this volume chronicles the development in 1976 of Tiny BASIC as an alternative to the "finger blistering," front-panel, machine-language programming which was then the only way to do things. This is always pertinent for the bit crunching and byte saving, language design theory, home-brew computer construction and the technical history of personal computing.
Topics include: Tiny BASIC, the (very) first word on CP/M, Speech Synthesis, Floating Point Routines, Timer Routines, Building an IMSAI, and more.

Vol. 2 1977 (Item #014)
1977 found DDJ still on the forefront. These issues offer refinements of Tiny BASIC, plus then state-of-the-art utilities, the advent of PILOT for microcomputers and a great deal of material centering around the Intel 6080, including a complete operating system. Products just becoming available for reviews were the H-8, KIM-1, MITS BASIC, Poly Basic, and NIBL.
Articles are about Lawrence Livermoore Lab's BASIC, Alpha Micro, String Handling, Cyphers, High Speed Interaction, I/O, Tiny Pilot & Turtle Graphics, many utilities, and even more.

Vol. 3 1978 (Item #015)
The microcomputer industry entered into its adolescence in 1978. This volume brings together the issues which began dealing with the 6502, with mass-market machines and languages to match. The authors began speaking more in terms of technique, rather than of specific implementations; because of this, they were able to continue laying the groundwork industry would follow. These articles relate very closely to what is generally available today.
Languages covered in depth were SAM76, Pilot, Pascal, and Lisp, in addition to RAM Testers, S-100 Bus Standard Proposal, Disassemblers, Editors, and much, much more.

Vol. 4 1979 (Item #016)
This volume heralds a wider interest in telecommunications, in algorithms, and in faster, more powerful utilities and languages, innovation is still present in every page, and more attention is paid to the best ways to use the processors which have proven longevity—primarily the 8080/Z80, 6502, and 6800. The subject matter is invaluable both as a learning tool and as a frequent source of reference.
Main subjects include: Programming Problems/Solutions, Pascal, Information Network Proposal, Floating Point Arithmetic, 8-bit to 16-bit Conversion, Psuedo-random Sequences, and Interfacing a Micro to a Mainframe—more than ever!

Vol. 5 1980 (Item #017)
All the ground-breaking issues from 1980 in one volume! Systems software reached a new level with the advent of CP/M, chronicled herein by Gary Kildall and others (DDJ's all-CP/M issue sold out within weeks of publication). Software portability became a subject of greater import, and DDJ published Ron Cain's immediately famous Small-C compiler—reprinted here in full.
Contents include: The Evolution of CP/M, a CP/M-Flavored C Interpreter, Ron Cain's C Compiler for the 8080. Further with Tiny BASIC, a Syntax-Oriented Compiler Writing Language, CP/M to UCSD Pascal File Conversion, Run-time Library for the Small-C Compiler and, as always, even more!

Vol. 6 1981 (Item #018)
1981 saw our first all-FORTH issue (now sold out), along with continuing coverage of CP/M, small-C, telecommunications, and new languages. Dave Cortesi opened "Dr. Dobb's Clinic" in 1981, beginning one of the magazine's most popular features.
Highlights: Information on PCNET, the Conference Tree, and The Electric Phone Book, writing your own compiler, a systems programming language, and Tiny BASIC for the 6809.

Vol. 7 1982 (Item #019)
In 1982 we introduced several significant pieces of software, including the RED text editor and the Runic extensible compiler, and we continue to publish utility programs and useful algorithms. Two new columns, The CP/M Exchange and The 16-Bit Software Toolbox, were launched, and we devoted special issues to FORTH and telecommunications. Resident Intern Dave Cortesi supplied a year of "Clinic" columns while delivering his famous review of JRT Pascal and writing the first serious technical comparison of CP/M-86 and MSDOS. This was also the year we began looking forward to today's generation of microprocessors and operating systems, publishing software for the Motorola 68000 and the Zilog Z8000 as well as Unix code. And in December, we looked beyond, in the provocative essay, "Fifth-generation Computers."

Complete your reference library. Buy the entire set of Dr. Dobb's Journals from 1976 through 1983, Bound Volumes 1–8, for $195.00. That's $34.00 off the combined individual prices—a savings of almost 15%!

To order by credit card, call toll free: 1-800-528-6050 ext. 4001. Refer to the item number of each volume. Or mail this coupon to: *Dr. Dobb's Journal, 2464 Embarcadero Way, Palo Alto, CA 94303.*

YES! ☐ Please send me the following Volumes of **Dr. Dobb's Journal.**
Payment must accompany your order.
Please charge my: ☐ Visa ☐ MasterCard ☐ American Express
I enclose ☐ Check/money order
Card # ________________ Expiration Date ________
Signature ________________
Name ________________ Address ________________
(please, no P.O. Boxes)
City ________ State ____ Zip ________
Mail to Dr. Dobb's Journal, 2464 Embarcadero Way, Palo Alto, CA 94303

Vol. 1	______	x $27.75 =	______
Vol. 2	______	x $27.75 =	______
Vol. 3	______	x $27.75 =	______
Vol. 4	______	x $27.75 =	______
Vol. 5	______	x $27.75 =	______
Vol. 6	______	x $27.75 =	______
Vol. 7	______	x $30.75 =	______
Vol. 8	______	x $31.75 =	______
All 8	______	x $195.00 =	______
		Sub-total $	______

California residents add applicable sales tax____% ______

Postage & Handling Must be Included with order.
Please add $1.75 per book in U.S. ($4.25 each surface mail outside U.S. Foreign Airmail rates available on request.)

TOTAL $ ______

Allow 6–9 weeks for delivery.

3110C

Listing Three

```
        OUTPUT FROM ZLKO TO CONSOLE DURING TEST OVERLAY CONSTRUCTION

->CMND(CTEST);
  CMND(
  CTEST)<--Command File Input
  ROOT(
  MAIN)
  ;
  OVINIT,<--Implied Include List
  OVRLAY,
  LØ;
  END;<--Searching For Required Modules

-- Building File on Disk --

Loading     (Ø:OVINIT .REL)
Loading     (Ø:OVRLAY  .REL)
Loading     (Ø:LØ      .REL)
Loading from(Ø:CRUNLIB .REL)

  -- Filename --                    -- File Information --
  Ø:MAIN    .COM        P.Base: Ø1ØØ   P.Size: ØEA7    Seg.End+1: 1182
                        D.Base: ØFBB   D.Size: Ø1C7
                        C.Base: 1182   C.Size: ØØØØ

  SYMFIL(
  CTEST)<--Creating Symbol File
  ;
  OVL(
  1,
  L1A)
  ;
  INCL(
  LIA)
```

(Continued on next page)

Listing Three

```
  OLV(
  1,
  L1B)
  ;
  INCL(
  L1B)
  ;
  END;<--Searching For Required Modules
  ;
-- Building File on Disk --

Loading     (0:L1B     .REL)
  -- Filename --                      -- File Information --
  0:L1B     .OVL        P.Base: 1182   P.Size: 000E    Seg.End+1: 11B7
                        D.Base: 119C   D.Size: 001B
                        C.Base: 11B7   C.Size: 0000

  OVL(
  1,
  L1C)
  ;
  INCL(
  L1C)
  ;
  END;<--Searching For Required Modules

-- Building File on Disk --

Loading     (0:L1C     .REL)

  -- Filename --                      -- File Information --
  0:L1C     .OVL        P.Base: 1182   P.Size: 00B2    Seg.End+1: 12F5
                        D.Base: 1240   D.Size: 00B5
                        C.Base: 12F5   C.Size: 0000

  OVL(
  2,
  L2A)
  ;
  INCL(
  L2A)
  ;
  END;<--Searching For Required Modules

-- Building File on Disk --

Loading     (0:L2A     .REL)

  -- Filename --                      -- File Information --
  0:L2A     .OVL        P.Base: 12F5   P.Size: 0045    Seg.End+1: 138C
                        D.Base: 1346   D.Size: 0046
                        C.Base: 138C   C.Size: 0000

  DOVL;
--- Overlay Structure ---

Name     Number   P.Base   End      Parent
----     ------   ------   ---      ------

L1A      0000     1182     11B6
L1B      0001     1182     11B6
L1C      0002     1182     12F4
L2A      0003     12F5     138B     L1C

DONE;
No Segment Under Construction

---------------------------------------------------------------------------
```

End Listings

WHO WOULD DIG THROUGH 300 DISKS OF MSDOS SOFTWARE TO FILL 14 DISKS FULL OF GEMS?

$12.00 STOCKING STUFFERS YOU'LL USE ALL YEAR!

DISK MS-1 ESSENTIAL UTILITIES

SWEEP is a powerful replacement for COPY. LU86403 makes (and unmakes) library files from your programs. (Bulletin boards contain many library files.) DSKRTN is a fancy disk utility which hides files, recovers files, and lists first sector of an erased file. WHEREIS finds file in subdirectories. SQUEEZE and UNSQUEEZE squeeze and expand files for faster and cheaper modem transfer. Plus WASH and several other file manipulators.

DISK MS-2 ASSEMBLER AND DISASSEMBLER

CHASM is compatible with the IBM macroassembler. ASMGEN is a disassembler whose output is compatible with the IBM macroassembler (works in 64K). MEMBRAIN converts extra RAM into a RAM disk. UNWORDSTAR converts Wordstar files to ASCII and vice versa. FSPOOL is a printer output to file redirector and SDIR is a super directory program. (And more . . .)

DISK MS-3 ADVENTURE

This is the original text-style adventure game that has kept whole throngs of (otherwise) serious programmers glued to their screens. It's ready to run but we also include source in CII and notes about how to expand this adventure (or create your own). Great fun for all computer users.

DISK MS-4 SINGLE USER RBBS

Turn your PC (or compatible) into a bulletin board and get called by the world. Includes source.

DISK MS-5 FILE AND DIRECTORY UTILITIES

UTIL sorts directories, moves files between directories, redefines the keyboard, and views files. Z sorts files. VTYPE is an easier to use TYPE. ST pages backwards or forwards through files. PROTECT AND UNPROTECT prevents accidental (or purposeful) erasure of files or programs. PCWINDOW a resident multi-function routine that handles notepads, timing, ASCII translation etc. Very handy Sidekick-like program.

DISK MS-6 FIG FORTH

(Includes MS-7, FORTH Source)

DISK MS-8 THREE GAMES

CASTLE is a graphic and conversational adventure. TREKRUN is a startrek style game. BOUNCING BABY is a finger exercise that will enthrall a 10-year-old.

DISK MS-9 PASCAL GAMES, PC MUSICIAN, CALCULATOR

This is our educational disk. PC MUSICIAN lets you write songs right on the screen and test them as you go. "I amazed my friends when I sat down at my computer and played . . ." The PASCAL GAMES aren't all that exciting, but the source should give you a good start on creating your own. PROGRAMMER'S CALCULATOR shows you how the stack, memory, and CPU registers work, plus it's highly calculating.

DISK MS-10 MODEM SOFTWARE

If you want to talk to bulletin boards this is your spot. QMODEM is the fanciest of the MODEM7 packages we've seen. It's powerful but easy to use. KERMIT is the standard package for communicating with mainframe (there are versions of KERMIT for nearly all the mainframes).

DISK MS-11 MULTI-USER RBBS 12.3 SYSTEM

Another complete RBBS, this one allows more than one user at a time.

DISK MS-12 SOURCE OF RBBS 12.3 SYSTEM

DISK MS-13 TEXT EDITOR & FORMATTER

ROFF4 is UNIX-like text formatter, PC-Write is one of the most famous of the freeware editors.

DISK MS-14 FORTH 84

For Super Fast Service Call 503-382-5060

MICRO CORNUCOPIA
The Micro Technical Journal

Disks $12.00 each Postpaid

MICRO CORNUCOPIA
P.O. BOX 223
BEND, OREGON 97709

MasterCard VISA

Circle no. 64 on reader service card.

ADVERTISER INDEX

* This advertiser prefers to be contacted directly: see ad for phone number.

Advertising Sales Offices

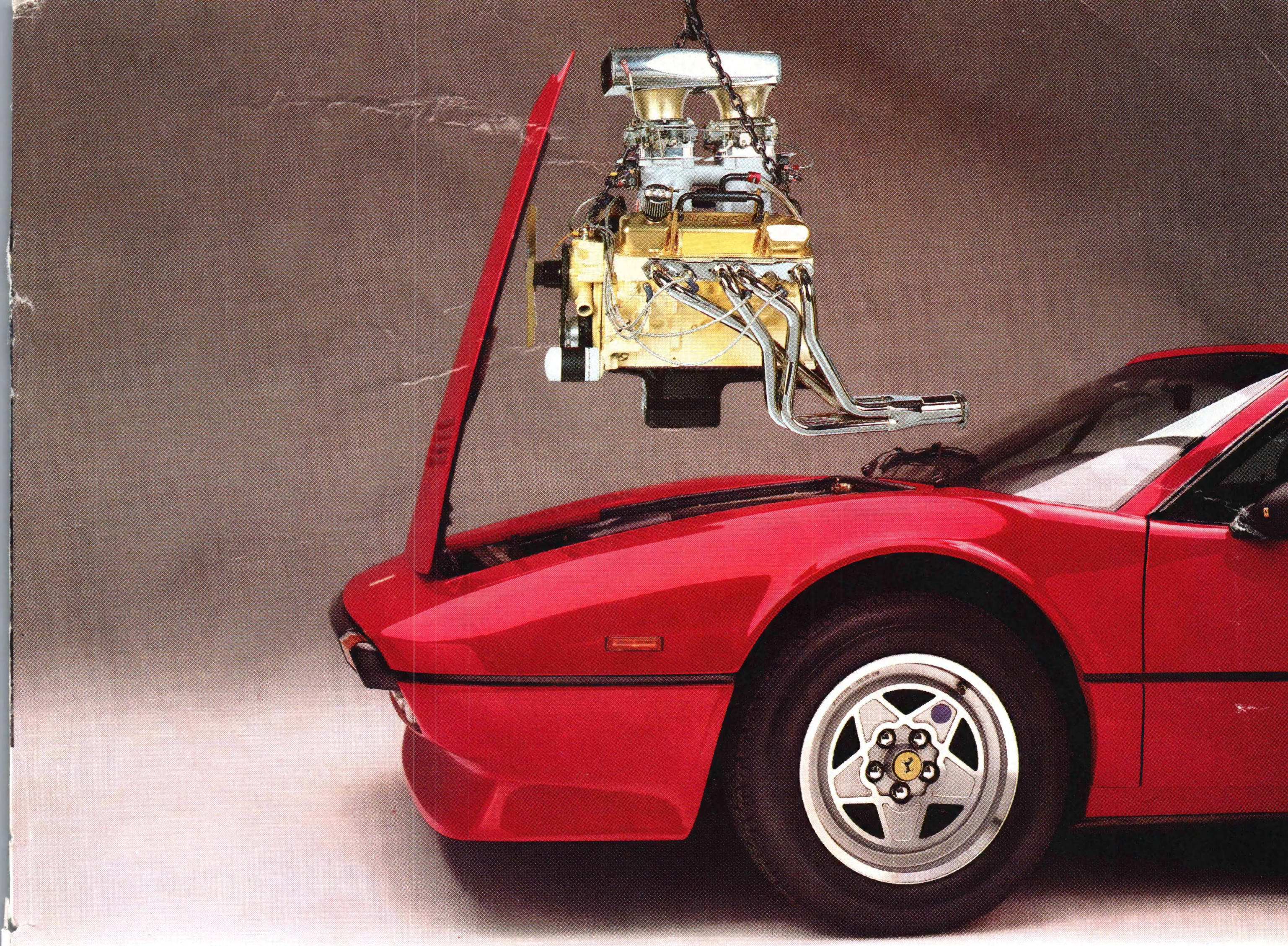
East Coast
Walter Andrzejewski (617) 567-8361

Midwest
Michele Beaty (317) 875-8093

Northern California/Northwest
Lisa Boudreau (415) 424-0600

Southern California/AZ/NM
Beth Dudas (714) 643-9439

Advertising Director
Shawn Horst (415) 424-0600



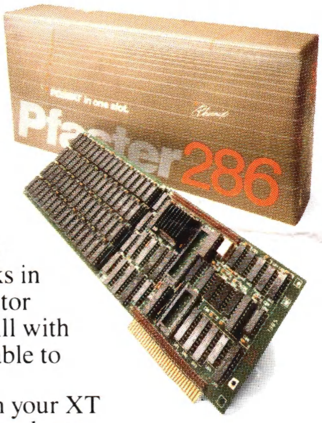
AT™ Pfantasies for your PC or XT.™

Want better speed and memory on your PC or XT without buying an AT?

You've got it!

Phoenix's new Pfaster™286 co-processor board turns your PC or XT into a high-speed engine 60 percent faster than an AT. Three times faster than an XT. It even supports PCs with third-party hard disks. But that's only the beginning.

You can handle spreadsheets and programs you never thought possible. Set up RAM disks in both 8088 and 80286 memory for linkage editor overlays or super-high-speed disk caching. All with Pfaster286's 1mb of standard RAM, expandable to 2mb, and dual-mode design.



You can develop 8086/186/286 software on your XT faster. Execute 95 percent of the application packages that run on the AT, excluding those that require fancy I/O capabilities your PC or XT hardware just isn't designed to handle. Queue multi-copy, multi-format print jobs for spooling. Or, switch to native 8088 mode to handle hardware-dependent programs and back again without rebooting. All with Pfaster286's compatible ROM software. And, Pfaster286 does the job unintrusively! No motherboard to exchange. No wires to solder. No chips to pull. Just plug it into a standard card slot, and type the magic word, "PFAST."

If you really didn't want an AT in the first place, just what it could do for you, call or write: Phoenix Computer Products Corp., 320 Norwood Park South, Norwood, MA 02062; (800) 344-7200. In Massachusetts, 617-762-5030.

Programmers' Pfantasies™
by
Phoenix

XT and AT are trademarks of International Business Machines Corporation. Pfaster286 and Programmers' Pfantasies are trademarks of Phoenix Computer Products Corporation.
For the Ferrari aficionado: yes, we know this is a rear engine car. We are showing the addition of a second engine to symbolize how Pfaster can be added to your PC or XT to increase performance.

Circle **no. 91** on reader service card.

SEE US AT FALL COMDEX

SAVE OVER 30% ON OUR GIFT PACKS!
60-DAY MONEY-BACK GUARANTEE

How Borland's Three New Holiday Packs Will Fill Your Stocking Without Emptying Your Piggybank.

Three special packs with dazzling discounts that will help get you into a Holiday mood. You can get some of Turbo, most of Turbo, or all of Turbo—including the two newest members of the Turbo family, Turbo GameWorks™ and Turbo Editor Toolbox.™ You also get our unmatched 60-day money-back guarantee, quality products that aren't copy-protected.

TURBO NEW PACK $95.00.

You get the two exciting new members of the Turbo Pascal family,

- TURBO GAMEWORKS, Chess, Bridge, and Go-Moku, complete with source code and a 200-page manual.
- TURBO EDITOR TOOLBOX, all the building blocks to make your own editors and word processors, complete with source code and a 200-page manual.

TURBO HOLIDAY PACK $125.00.

You get all three of the Turbo family classics for only $125.00 (about a 30% discount). Turbo Pascal 3.0 and Turbo Tutor and Turbo Data-Base Toolbox—all for just $125.00.

- TURBO PASCAL combines the fastest Pascal compiler with an integrated development environment.
- TURBO TUTOR teaches you step-by-step how to use Turbo Pascal with commented source code for all program examples on diskette.
- TURBO DATABASE TOOLBOX offers three problem-solving modules for your Turbo Pascal programs: Turbo Access, Turbo Sort, and GINST, which generates a ready-to-run installation program that lets you forget about adapting your software to specific terminals.

TURBO HOLIDAY JUMBO PACK $245.00.

This is it—the whole thing, the entire Turbo family including its two newest members. You get:

- Turbo Pascal
- Turbo Graphix Toolbox
- Turbo Tutor
- Turbo DataBase Toolbox
- Turbo GameWorks NEW!
- Turbo Editor Toolbox

and you pay only $245.00 for all six! Which means that you're getting everything at only about $40 a piece. Quite a holiday deal. (And if you already own one or several members of the Turbo family, be creative—nothing can stop you from buying the Jumbo Pack, picking out the ones you already have and giving the rest as holiday gifts to family and friends. At these prices you can afford to give to others and to yourself.) Speaking of Holidays, this offer lasts until March 31, 1986. (At Borland, we like to make the Holidays last.)

NEW! TURBO GAMEWORKS $69.95.

Our new Turbo GameWorks offers games you can play and replay without Turbo Pascal or revise and rewrite with Turbo Pascal 3.0. We give you the source code, the manual, the diskettes and the competitive edge. Chess, Bridge and Go-Moku. State-of-the-art games that let you be player, referee, and rules committee all at once because you have the Turbo Pascal source code. Learn exactly how the games are made—so you can go off and make your own. And Turbo GameWorks is the only quality game you can buy that is not copy-protected. Sold separately, only $69.95. (Just $47.50 if you buy the Turbo New Pack.)

NEW! TURBO EDITOR TOOLBOX $69.95.

Build your own word processor—for only $69.95! You get ready-to-compile source code, a full-featured word processor that looks and acts like WordStar™ and a 200-page manual that tells you how to integrate the editor procedures and functions into your programs. With Turbo Editor Toolbox you can have the best of all word processors. You can make WordStar behave like Multi-Mate. Support windows just like Microsoft's Word. And do it as fast as WordPerfect does it. Incorporate your new "hybrids" into your programs to achieve incredible control and power. Sold separately, only $69.95. (If you buy the Turbo New Pack, the price drops to just $47.50.)

NOT COPY-PROTECTED

Holiday Gift Packs, Turbo GameWorks™ & Turbo Editor Toolbox™

Available at better dealers nationwide. Call (800) 556-2283 for the dealer nearest you. To order by Credit Card call (800) 255-8008, CA (800) 742-1133.

		Quantity
*Turbo Holiday Jumbo Pack	$245.00	___
*Turbo Holiday Pack	$125.00	___
*Turbo New Pack	$95.00	___
Pascal	$69.95	___
Pascal w/8087	$109.90	___
Pascal w/BCD	$109.90	___
Pascal w/8087 and BCD	$124.95	___
Turbo DataBase	$54.95	___
Turbo Graphix	$54.95	___
Turbo Tutor	$34.95	___
Turbo Editor	$69.95	___
Turbo GameWorks	$69.95	___

These prices include shipping to all U.S. cities. All foreign orders add $10 per product ordered.

*Gift Pack Offers Last Until March 31, 1986

The holiday packs include an upgrade coupon for both options so you get BCD and 8087 support for $39.95 (regularly $55.00).

Carefully describe your computer system!

Mine is: __ 8-bit __ 16-bit

I use: __ PC-DOS __ MS-DOS __ CP/M-80 __ CP/M-86

My computer's name and model is: ______

The disk size I use is: ☐ 3½" ☐ 5¼" ☐ 8"

Name: ______

Shipping Address: ______

City: ______

State: ______ Zip: ______

Telephone: ______

H11

Amount: (CA add 6% tax) ______

Payment: VISA MC Check Bank Draft

Credit Card Expiration Date: ______ / ______

Card # ______

NOTE: Turbo Editor Toolbox and Turbo GameWorks are available for the IBM PC and true-compatibles using Turbo Pascal 3.0 ONLY.

COD's and Purchase Orders will not be accepted by Borland International. California residents: add 6% sales tax. Outside USA: add $10 and make payment by bank draft, payable in U.S. dollars drawn on a U.S. bank.

BORLAND INTERNATIONAL

4585 Scotts Valley Drive, Scotts Valley CA 95066
Phone (408) 438-8400 Telex 172373

Circle no. 161 on reader service card.